# Chronos: A Tool for Handling Temporal Ontologies in Protégé

Alexandros Preventis

Department of Electronics and Computer Engineering
Technical University of Crete

Dissertation Thesis Committee:
Euripides G.M. Petrakis, Associate Professor (Supervisor)
Minos Garofalakis, Professor
Michail G. Lagoudakis, Assistant Professor

2011

**Acknowledgments**

This thesis would not have been possible without the help of several people who in one way or another, contributed and offered their valuable assistance in the preparation and completion of this study.

First and foremost, my utmost gratitude to my advisor, Associate Professor Euripides G. M. Petrakis for his supervision, advise and guidance from the very early stage of this research. I am grateful for his encouragement and precious contribution throughout this study. I would also like to thank him for giving me the opportunity to work on this very interesting field of research.

Also, I would like to thank Professor Minos Garofalakis and Assistant Professor Michail G. Lagoudakis who agreed to evaluate my diploma thesis.

Moreover, I would like to thank my laboratory colleagues for their patient and constructive comments.

I would like to thank all my friends for these great years we spent together and for the many wonderful memories.

Most of all, I would like to thank my family for their enormous help, understanding and support throughout all these years as a student.

**Abstract**

The Semantic Web is the extension of the World Wide Web that makes the information machine-readable by using sets of concepts known as ontologies. An ontology describes the concepts in a domain of interest and also the relationships that hold between those concepts. The Web Ontology Language OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is biased towards binary relations. Notice that, relations that vary in time are in fact ternary. The representation of ternary relations is a known problem whose solution calls for methods referred to as 4D-fluents [4] or N-ary relations [5], with the later being a W3C recommendation. Apart from this, the two approaches are rather equivalent so that, selecting either one is simply a matter of taste. However, the resulting ontologies (with both approaches) are rather complicated and difficult to read or handle. Facilitating crafting (i.e., creating, editing) of temporal ontologies in Protege [6] (a common OWL editor) is exactly the problem this work is dealing with. In this thesis we introduce *Chronos*, a plug-in for Protégé OWL editor (version 4.1) that facilitates creation and editing of temporal ontologies. Chronos does not require the user to be familiar with the peculiarities of the underlying representation model. Particular emphasis has been given in the maintenance of the property semantics and restrictions, even after a property has been converted from static to temporal.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The Semantic Web is the extension of the World Wide Web that enables people to share content beyond the boundaries of applications and websites. It makes the information machine-readable by using a formal representation of knowledge as a set of concepts, called ontology.

An ontology describes the concepts in a domain of interest and also the relationships that hold between those concepts. Many standard ontology languages have been developed, the most recent of which is OWL [1]. OWL enables greater machine interpretability of Web content than XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. It is based on a logical model that allows to reason about the entities and check whether or not all the statements and definitions described in the ontology are mutually consistent. The basic elements of an OWL ontology are classes, properties, instances of classes (individuals) and relationships between these instances [2].

Properties in OWL are distinguished in two types, data properties which relate an individual to an XML Schema data type and object properties, which are relations on individuals. Both property types are binary. This feature of OWL makes representation of relations that change over time (temporal relations) a difficult task to deal with [4]. A temporal relation is represented as a ternary relation between two instances and the interval during which the relation holds, which is not permitted in OWL.

## 1.1 Problem Definition

The problem of representing temporal relationships in a binary-limited representation is not new. Approaches to deal with this problem do exist (e.g., 4D-fluents or N-ary relations) but all suffer from significant drawbacks (data redundancy, object proliferation, limited reasoning support). In addition, all, result in complicated ontologies compared with their static counterparts where all relations do not change in time (as every temporal relation is

substituted by a set of binary OWL relations) [4].

OWL ontology editors such as Protégé [6] provide a well-suited environment allowing users to create or edit static OWL ontologies with binary relations. Yet, they provide no means for facilitating representation and crafting of temporal ontologies with ternary relations. As it is common in all known approaches for representing dynamic concepts (such as N-ary relations, 4D-Fluents), ternary relations are decomposed into sets of binary relationships. Those sets of binary relationships do not hold between entities of the ontology as they used to, but they now refer to entities introduced by the temporal representation model adopted.

The creation of a temporal ontology, or the conversion of a static ontology to temporal, even with the use of ontology editors, can be a very complex, time-consuming and error-prone procedure. The user must have very good knowledge of the selected representation model and all of its peculiarities. This is exactly the problem that this work is dealing with.

## 1.2 Proposed Solution

We introduce *Chronos*, a plug-in for Protégé OWL editor (version 4.1) that facilitates creation and editing of temporal ontologies. Through Chronos, the user can add the dimension of time in an ontology. Chronos enables handling of temporal relations in Protégé the same way static ones are handled. The representation model used for the creation of temporal relationships is the N-ary model [5]. The user does not have to be familiar with the peculiarities of the underlying model, as the instances and the relationships between them that need to be added for the representation of the ternary relations are handed by our tool. Chronos is implemented as a Tab plug-in for Protégé, allowing the user to handle both static and temporal ontologies.

The contribution of this work is the development of Chronos, a tab widget plug-in for Protégé 4.1, which has the following desirable features:

1. Chronos is easy to use, handles temporal ontologies as the static ones, allowing the user not to be familiar with the peculiarities of the underlying temporal representation model.

2. Supports restriction adding and checking both, on temporal properties (e.g., an employee cannot work for two different companies at the same time), classes (e.g., a company cannot employ more than 20 employees at the same time) and on individuals.

3. Supports reasoning over the temporal ontologies using the standard Pellet reasoner [7] for Protégé. Temporal reasoning is realized by a set of SWRL rules [8] [9, 10] which provide reasoning capabilities over time intervals.

4. Chronos interface is consistent with the layout of the default Protégé tabs.

## 1.3 Thesis outline

Background knowledge and related work are discussed in Chapter 2. This includes description of OWL and SWRL, the N-ary relations representation model employed in Chronos and the Protégé OWL editor. The functionalities of Chronos will be discussed in Section 3. More specifically, issues related to the representation of temporal relations are discussed in Section 3.1. Our approach for moving from static to dynamic ontologies is presented in Section 3.2 and Section 3.3 shows how we deal with property restrictions. Chapter 4 presents Chronos' interface, followed by use cases, activity diagrams and the plug-in documentation. Finally, conclusions and issues for further research are discussed in Chapter 5.

# Chapter 2

# Background and Related Work

## 2.1 Ontologies

There have been several attempts to define the word "ontology" in computer science. Some of them approach "ontology" from a more philosophical scope, while some others have tried to give a more practical definition of the term. In our work, we go with the practical definition, that an ontology is a method of representing items of knowledge (ideas, facts, things—whatever) in a way that defines the relationships and classifications of concepts within a specified domain of knowledge [11]. The four fundamental ingredients of an ontology are classes, properties, individuals and datatypes.

**Individuals** represent objects in the domain of interest.

**Properties** represent relationships. They are distinguished in two types, *Object properties* and *Data properties*. Object properties relate two individuals to each other and data properties relate an individual to a data value.

**Classes** represent sets of individuals (or sets of objects) that share some common features, constraints and semantics.

Ontologies provide the semantic foundation, that allows agents and their human users to identify, collect and process suitable information by interpreting the semantic metadata based on the task at hand. They also allow the exchange of results and the communication by sharing such resources [12].

## 2.2 OWL

The Web Ontology Language OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web. It is developed as a

vocabulary extension of RDF (the Resource Description Framework) and is derived from the DAML+OIL Web Ontology Language. OWL is intended to provide a language that can be used to describe the classes and relations between them that are inherent in Web documents and applications. OWL provides three increasingly expressive sublanguages:

1. OWL Lite supports classification hierarchy and simple constraints.
2. OWL DL supports maximum expressiveness without loosing computational completeness and decidability.
3. OWL Full provides maximum expressiveness and the syntactic freedom of RDF, but without computational guarantees.

Each of these sublanguages is an extension of it's simpler predecessor.

OWL is part of W3C's Semantic Web technology stack. In October 2007 a new W3C working group was started to extend OWL with several new features. This new version was called OWL 2 [13].

## 2.3 SWRL

Semantic Web Rule Language (SWRL) [8] is a proposal for a semantic web rules-language, combining sublanguages of OWL (OWL DL and OWL Lite) and the Rule Markup Language. SWRL allows for rule expressions involving OWL concepts enabling more powerful deductive reasoning than OWL alone. Semantically, SWRL is built on the same description logic foundation as OWL and provides similar strong formal guarantees of inference.

A SWRL rule contains an antecedent part, which is referred to as the body, and a consequent part, which is referred to as the head. Both the body and head consist of positive conjunctions of atoms, meaning that all atoms of the antecedent part must be satisfied in order for the consequent to be applied.

## 2.4 OWL-Time Ontology

OWL-Time [3] is an OWL ontology of temporal concepts, for describing the temporal contents of Web pages and the temporal properties of Web services. It provides a vocabulary for describing topological relations between temporal entities, such as instants or intervals, together with information about durations. Allen [14] has developed a calculus of binary relations for representing qualitative temporal information on intervals that addresses the problem of reasoning over such information. OWL-Time also provides these relations on the intervals.

## 2.5 Temporal Representation

Dealing with information that changes over time is a critical problem for practical Knowledge Representation languages (OWL, RDF). These language are based on binary relationships between concepts. In the following, we discuss some of the representation models that we considered to use in Chronos, along with their advantages and disadvantages. Figure 2.1 illustrates an example of a ternary relationship. Its representation using various models is discussed next.

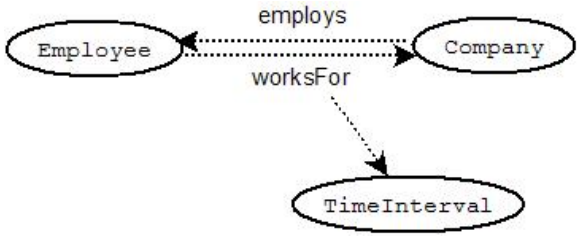


Figure 2.1: A relationship that holds during a specific time interval, represented as a ternary relationship.

**Temporal Description Logics (TDLs)** [15, 16] extend the standard description logics (DLs) that form the basis of Semantic Web standards with additional constructs such as *'until'* and *'always in the past'*. The TDLs restrict the OWL operators in order to keep all the reasoning problems in the language decidable. Most importantly for our work, TDLs cannot support quantitative information, making it not possible to use the full Allens calculus.

**Versioning** [17] suggests that an ontology has different versions (one per instance of time). This method suffers from several disadvantages, as a simple change on an attribute of the ontology requires the creation of a new version of the ontology. This results to huge information redundancy. Moreover, searching for an event that occurred at a specific time, requires exhaustive search between the different ontology versions. Lastly, it is not clear how relations between evolving classes is represented.

**Reification** is a general purpose technique for representing n-ary relations using languages that only permit binary relations. An n-ary relation is represented as an object, which is the subject of n+1 triples. Those triples have as objects the participants of the n-ary relation plus one triple has as object the predicate of the property. In our example, supposing that we want to represent the statement *"John has worked for Apple from 2000 to 2005"* is expressed as *livesIn(John, Apple, t)*

where $t$ represents the interval '2000 to 2005'. Using reification, this relationship is represented as a new object with *John*, *Apple*, $t$ and *worksFor* being objects of properties (Figure 2.2). Reification also suffers from data redundancy, because a new object is created for every reified relationship. This is a common problem to all approaches that are not based on non temporal Description Logics. A major disadvantage of reification is that it offers limited reasoning capabilities. This is a result of representing the predicate of the relation as an object of a property thus the OWL semantics over the property are no longer applicable.

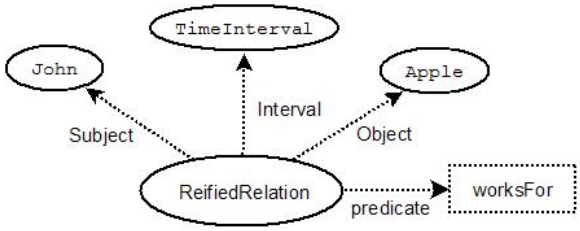


Figure 2.2: Example of Reification.

**4D Fluents (Four-Dimensional Approach)** [4] considers entities to exist in time the same way that material objects exist in space—they occupy spacetime. These entities have temporal parts (time slices) that represent the entity during a time interval. Therefore, according to this approach, a temporal property does not hold between the static entities but between their temporal parts. The time slices of an entity have a specific lifetime, that is the time interval of the relation that they participate in. Thus, a time slice can participate in more than one temporal relations (fluents) only if they hold for precisely the same time interval.

As shown in Figure 2.3 for every time interval during which the property *worksFor* holds, two instances of the class *TimeSlice* are created. These instances are related to the time interval that defines their "lifetime", to the static entities whose they are temporal parts of and to each other with the fluent property *worksFor*.

The 4D-fluents approach suffers from data redundancy, since two individuals and four properties between them have to be added to the ontology. It also complicates the ontology. Nevertheless, it is a sufficient way to represent relations that evolve over time, maintaining the property semantics and offering reasoning capabilities.

**N-ary Relations** [5] is a general purpose technique for representing n-ary relations. In N-ary Relations representation model, the static entities

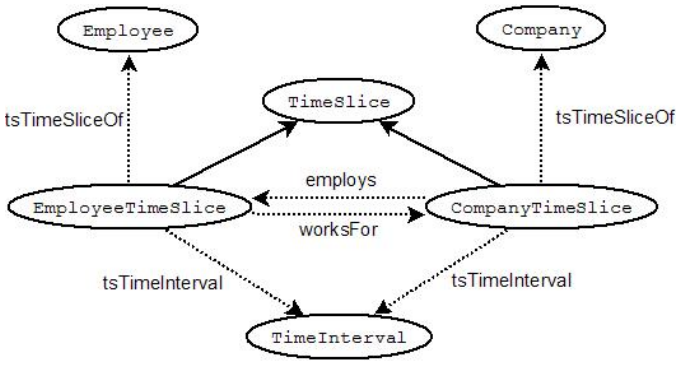


Figure 2.3: Example of the 4D Fluents approach. The dotted arrows represent object properties between classes. The solid arrows represent the 'isA' relationship.

are considered to participate in events. A temporal property property between two individuals (e.g. Employee works for Company) holds as long as that event endures. The n-ary property is represented as a class rather than as a property. Instances of such classes correspond to instances of the relation. Additional properties provide binary links to each argument of the relation. In contrast to reification, the n-ary relation is not represented as the object of a property but as two properties each related with the new object.

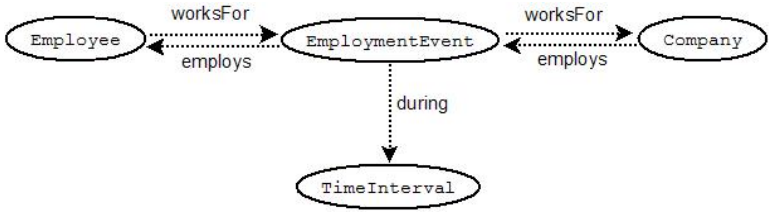


Figure 2.4: Example of the N-ary Relations representation.

As shown in Figure 2.4, for the representation of properties that change over time the property's domain and range are modified, in order to connect to the new object that is added. The new domain is the union of the old domain with the class that represents the n-ary property (*Event* class). Likewise, the new range is a union of the old one with the *Event* class.

The information redundancy using N-ary Relations is minimal in comparison to the other methods described (just one object and two properties have to be added to the ontology for each temporal triple). Property semantics are maintained and it also offers reasoning capabilities. Those reasons, along with the fact that the particular model

is a W3C recommendation for representing n-ary relations, have led us to use this model in our work.

## 2.6 Protégé OWL Editor

Protégé [6] is a free, open-source platform that provides a suite of tools for building domain models and knowledge-based applications with ontologies. It supports the creation, visualization and manipulation of ontologies in various representation formats. Protégé can be customized to provide domain-friendly support for creating knowledge models and entering data. Further, Protégé can be extended by way of a plug-in architecture and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications. The Protégé platform supports two main ways of modeling ontologies, the Protégé-Frames editor and the Protégé OWL editor.

The Protégé-OWL editor enables users to build ontologies for the Semantic Web, in particular in the W3C's Web Ontology Language (OWL) [2]. Version 4.1 of Protégé OWL provides full support of OWL 2.0 [13]. This is the version which Chronos plug-in extends.

## 2.7 OWL API

The OWL API [18] is a Java interface and implementation for the W3C Web Ontology Language (OWL), used to represent Semantic Web ontologies. OWL API has been designed to meet the needs of people developing OWL based applications, OWL editors, such as Protégé and OWL reasoners. It is a high level API that is focused towards the OWL DL and OWL 2 specifications and supports ontology management, ontology change, ontology parsing and rendering, data structure storage and reasoner interfaces.

# Chapter 3

# Chronos Tab Plug-in for Protégé

Chronos is a tab plug-in for Protégé 4, that facilitates the creation and editing of temporal ontologies. It is compatible with the OWL DL and OWL 2.0 specifications. With the addition of a set of SWRL rules, created by Batsakis et al. [9, 10], it supports reasoning capabilities and appliance of temporal restrictions by using the build-in Pellet reasoner [7]. Static object and data properties can be converted to temporal easily. New temporal object or data properties can be added between individuals, that hold during a specific or non-specific (qualitative) time interval. Cardinality constraints and property restrictions on temporal properties can also be handled with our tool (as described in section 3.3).

Chronos makes it possible to convert a static ontology to temporal, or combined with Protégé's standard tabs, to create a temporal ontology from scratch. The user does not get involved with the confusing details of the underlying model, as all the intermediate objects and the relationships between them are handed by Chronos. Anyone who is familiar with Protégé editor and OWL language can use Chronos to add the dimension of time to the ontologies he develops.

## 3.1 Handling Temporal Ontologies in Protégé

Protégé OWL is a platform that provides all the tools necessary for crafting OWL ontologies. Although it is feasible to create a temporal ontology using the standard Protégé tabs, it is a very complex and time-waisting procedure. As described in section 2.5, a lot of changes have to take place in an ontology in order to convert *just one* property from static to temporal. The property's domain and range have to be modified, new intermediate classes and objects have to be added and new properties between them. The changes that are introduced by the temporal representation model (the N-ary Relations model

in our case) have to be applied to all the objects that are connected through this property. All the class expressions and restrictions that refer to the converted property have to adjust to the changes made as well.

Chronos plug-in is designed to enable the user create temporal ontologies, or edit existing ones as they were static. The user does not have to get involved with intermediate objects or relationships between them. All the peculiarities of the model that may be complex and confusing are hidden by the user, keeping the ontology simple.

## 3.2 Moving from Static to Dynamic

The conversion of an ontology from static to dynamic requires a lot of changes. Different types of entities (e.g., object properties, data properties) are handled differently. Particular emphasis is given on maintaining the semantics of those entities, after they are converted to temporal.

In order to implement the changes suggested by the N-ary Relations model [5] the following new objects are introduced into the ontology.

- *Event*, the class that represents the n-ary property.
- *during*, an object property that relates the event to the time interval during which it holds.
- *participatesIn*, an object property that relates the individuals that participate in an event, to that specific event individual. The object properties that are converted to temporal become sub-properties of this property.
- *overlaps*, an object property that relates two time intervals. This property implies that those time intervals, in some way overlap to each other.

In the following subsections, we describe the way that Chronos handles the different kinds of OWL entities, using the specifications introduced by the N-ary Relations model.

### 3.2.1 Object Properties

The representation of a temporal object property, according to the N-ary Relations representation model, suggests that an intermediate object (instance of the *Event* class) is introduced between the subject and the object of the static object property. This object appears as both, an object and a subject in two triples, whose predicate is the specific object property, and together represent the temporal relationship. To make this possible, the static property's domain and range have to be modified. The dynamic property's domain/range will be the union of the static property's domain/range and

the *Event* class. Moreover, the converted object property is made a sub-property of the *participatesIn* property. Lastly, the *Event* class is related to the time interval class(*Interval*) with the object property *during*.

In our example (Figure 2.4), the domain of the static object property *worksFor* is class *Employee*. After conversion, the domain of the dynamic object property will be the anonymous class *(Employee OR Event)*, that represents the union of the classes *Event* and *Employee*. The static property's range is modified respectively, from *Company* to *(Company OR Event)*.

### 3.2.2 Data Properties

Data properties are handled by Chronos in a similar way to object properties. The dynamic data property's domain is the union of the static property's domain and the *Event* class. The main difference with the object properties is that the range cannot be the union of a data type and the *Event* class[1]. To overcome this problem, we create an object property named by the data property, and followed by "OP". This object property will relate the static data property's domain to the *Event* class. This is also made a sub-property of the *participatesIn* object property. The data property with the modified domain, will connect the event to the data type. As in object property conversion, the *Event* is related to the *Interval* with the *during* object property.

An example of a converted data property is illustrated in Figure 3.1. The examples shows how would the temporal data property *hasPrice* be represented, according to our approach.

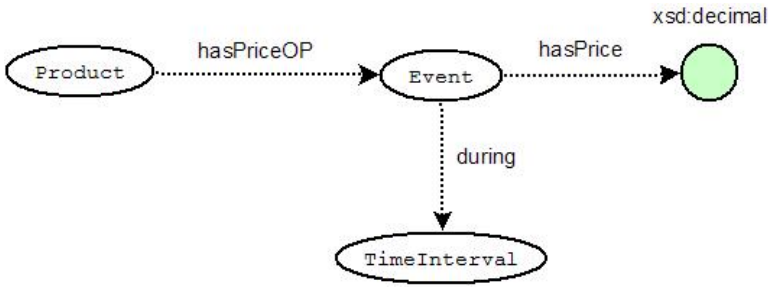


Figure 3.1: Example of a temporal data property using N-ary Relations.

[1]In OWL DL there is a concrete distinction between OWL Data Types and OWL Classes. OWL Full, on the other hand, allows the union of data types and classes. The Proteégé editor and the OWL API though, support the OWL DL expressiveness, not the OWL Full.

### 3.2.3 Individuals

Individuals represent objects in the domain of interest. For example, 'John' is an individual of the class 'Person'. The statement *"John has lived from 1920 to 1998"* does not require a temporal property for its representation. However, when a property evolves over time, such as in the statement *"John has lived in Athens from 1950 to 1985"*, the property *livesIn* is a temporal property that holds during a specific time interval. John still lived after the year 1985, but in a different place. In this case, temporal relations are defined in terms of relations between individuals rather than temporal individuals (i.e., temporal properties).

When a property is converted to temporal, all the triples that contain this property are converted too. For each triple in the ontology, a new instance of the *Event* class is created and introduced between the subject and the object, as explained in subsections 3.2.1 and 3.2.2. This event individual is connected to a *TimeInterval* instance with the *during* object property. The *TimeInterval* individual is related to two *Instant* individuals, one that represents the starting point of the interval and one that represents the ending point of the interval. Each of these *Instant* individuals are connected to a *dateTime* data type with the data property *inXSDDateTime*. Figure 3.2 illustrates the individuals and the relationships between them for a temporal object property.

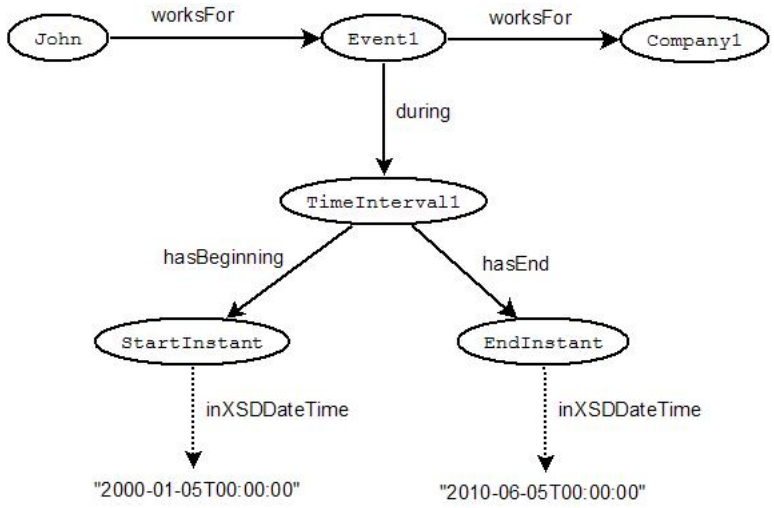


Figure 3.2: Example of a temporal object property between individuals.

It is now clear that the creation of a temporal ontology using the conventional Proteégé tabs proves to be a very difficult task. Chronos enables the user to create a new individual related to an other with a temporal property, add temporal object or data property assertions to already existing individuals, or edit the time intervals during which the temporal property

assertions hold. The user does not have to intervene with the intermediate objects or with their relationships, making the manipulation of temporal relationships between individuals as easy as the manipulation of the static ones.

### 3.2.4 Classes

Classes provide an abstraction mechanism for grouping resources with similar characteristics. Every OWL class is associated with a set of individuals, called the class extension. The individuals in the class extension are called the instances of the class. A class has an intensional meaning (the underlying concept) which is related but not equal to its class extension. Thus, two classes may have the same class extension, but still be different classes.

In the same way to individuals, classes cannot be converted to temporal. So, when we use the term "convert a class", we refer to the object and data properties that relate to this class. More specifically, when the user converts a class to temporal using Chronos, the entities that will be affected are: (a) the object and data properties that relate members of the selected class, (b) the object and data properties where this class appears as a *Domain* and (c) the restrictions where one of these object or data properties appear in.

It is easily concluded that the "class conversion" does not provide any additional functionality to our tool. It is just a convenient way to convert multiple object and data properties to temporal.

## 3.3 Dealing with Cardinality Constraints and Property Restrictions

OWL classes are described through *'class descriptions'*. A class description is the term used in this document for the basic building blocks of class axioms (informally called class definitions). A class description describes an OWL class, either by a class name or by specifying the class extension of an unnamed anonymous class.

OWL distinguishes six types of class descriptions:

1. A class identifier (a URI reference).
2. An exhaustive enumeration of individuals that together form the instances of a class.
3. A property restriction.
4. The intersection of two or more class descriptions.
5. The union of two or more class descriptions.
6. The complement of a class description.

A property restriction is a special kind of class description. It describes an anonymous class, namely a class of all individuals that satisfy the restriction. OWL distinguishes two kinds of property restrictions: value constraints and cardinality constraints.

Chronos enables the addition of temporal constraints to ontologies. A temporal constraint can be added to the active ontology either as a complex class description, or as a SWRL rule. In the first case, the constraint can be applied as a *necessary and sufficient condition*. In the case of a SWRL rule constraint though, it can only be applied as a *necessary condition*.

All the definitions given in this section are taken from the *"OWL Web Ontology Language Reference"* [19].

### 3.3.1 Value Constraints

#### 3.3.1.1 owl:allValuesFrom

The value constraint owl:allValuesFrom is a built-in OWL property that links a restriction class to either a class description or a data range. A restriction containing an *owl:allValuesFrom* constraint is used to describe a class of all individuals for which all values of the property under consideration are either members of the class extension of the class description or are data values within the specified data range. In other words, it defines a class of individuals $x$ for which holds that if the pair *(x,y)* is an instance of $P$ (the property concerned), then $y$ should be an instance of the class description or a value in the data range, respectively.

In the case that the property concerned is temporal, the form of the constraint is different. The restriction is used to describe a class of all individuals for which all values of the property under consideration are those members of the *Event* class that are connected with the concerned property to either members of the class extension of the class description or are data values within the specific data range.

For example, a restriction on a class 'Company' (in Manchester OWL syntax [20]) could be

*employs* ***only*** *Employee*

If the object property *employs* is temporal, the restriction becomes:

*employs* ***only*** *(Event* ***and*** *(employs* ***only*** *Employee))*

#### 3.3.1.2 owl:someValuesFrom

The value constraint *owl:someValuesFrom* is a built-in OWL property that links a restriction class to a class description or a data range. A restriction

containing an *owl:someValuesFrom* constraint describes a class of all individuals for which at least one value of the property concerned is an instance of the class description or a data value in the data range. In other words, it defines a class of individuals $x$ for which there is at least one $y$ (either an instance of the class description or value of the data range) such that the pair *(x,y)* is an instance of P. This does not exclude that there are other instances *(x,y')* of $P$ for which $y$' does not belong to the class description or data range.

The restriction, in the case that a temporal property is concerned, is used to describe a class of all individuals for which at least one value of the property concerned is an *Event* individual that is connected with the concerned property to an instance of the class description or a data value in the data range.

For example, a restriction on a class 'Company' could be:

*employs* ***some*** *Employee*

If the object property *employs* is temporal, the restriction becomes:

*employs* ***some*** *(Event* ***and*** *(employs* ***some*** *Employee))*

#### 3.3.1.3 owl:hasValue

The value constraint *owl:hasValue* is a built-in OWL property that links a restriction class to a value $V$, which can be either an individual or a data value. A restriction containing a *owl:hasValue* constraint describes a class of all individuals for which the property concerned has at least one value *semantically equal* to $V$ (it may have other values as well).

Our approach for the temporal form of that restriction is that it describes a class of all individuals for which the property concerned has at least one value *semantically equal* to $V$, for each event that these individuals participate in.

This temporal constraint is applied with the addition of an SWRL rule that to the ontology. For example, an owl:hasValue restriction on a class 'Company' is:

*employs* ***value*** *John*

where 'John' is an individual of the class Employee. The SWRL rule that would be added to the ontology to apply the temporal constraint would be:

$$Company(?x) \wedge participatesIn(?x,\ ?e) \wedge Event(?e) \rightarrow employs(?e,\ John) \wedge employs(?x,\ ?e)$$

meaning that for each event that an individual of the class 'Company' participates in, that company individual is also related to the Employee 'John' with the temporal object property *employs*.

### 3.3.2 Cardinality Constraints

In OWL, like in RDF, it is assumed that any instance of a class may have an arbitrary number (zero or more) of values for a particular property. To make a property required (at least one), to allow only a specific number of values for that property, or to insist that a property must not occur, cardinality constraints can be used. OWL provides three constructs for restricting the cardinality of properties locally within a class context.

#### 3.3.2.1 owl:maxCardinality

The cardinality constraint *owl:maxCardinality* is a built-in OWL property that links a restriction class to a data value belonging to the value space of the XML Schema datatype nonNegativeInteger. A restriction containing an *owl:maxCardinality* constraint describes a class of all individuals that have at most $N$ semantically distinct values (individuals or data values) for the property concerned, where $N$ is the value of the cardinality constraint. Syntactically, the cardinality constraint is represented as an RDF property element with the corresponding rdf:datatype attribute.

Our approach for the *maxCardinality* constraint when concerning temporal properties is that it is a constraint that describes a class of all individuals that have at most $N$ semantically distinct values *at the same time*, for the property concerned. This temporal constraint is applied with the addition of a SWRL rule to the ontology. For example, the *owl:maxCardinality* constraint in Manchester syntax is:

*employs* ***max 2*** *Employee*

It implies that an individual of the class 'Company' cannot be related to more than two individuals of the class Employee with the object property *employs*. The SWRL rule that would be added to the ontology to apply the temporal version of the constraint would be:

*Event(?e0)* $\wedge$ *Event(?e1)* $\wedge$ *Event(?e2)* $\wedge$ *Company(?x)* $\wedge$ *Employee(?y0)*
$\wedge$ *Employee(?y1)* $\wedge$ *Employee(?y2)* $\wedge$ *during(?e0, ?i0)* $\wedge$ *during(?e1, ?i1)*
$\wedge$ *during(?e2, ?i2)* $\wedge$ *overlaps(?i0, ?i1)* $\wedge$ *overlaps(?i0, ?i2)*
$\wedge$ *overlaps(?i1, ?i2)* $\wedge$ *employs(?e0, ?y0)* $\wedge$ *employs(?e1, ?y1)*
$\wedge$ *employs(?e2, ?y2)* $\wedge$ *employs(?x, ?e0)* $\wedge$ *employs(?x, ?e1)*
$\wedge$ *employs(?x, ?e2)* $\wedge$ *DifferentFrom(?y0, ?y1)*
$\wedge$ *DifferentFrom(?y0, ?y2)* $\wedge$ *DifferentFrom(?y1, ?y2)*
$\rightarrow$ *Nothing(?x)*

This means that if there are three *different* individuals of the class 'Employee' that relate through the temporal property *employs* to the same individual of the class 'Company', and the time intervals associated with those temporal properties pairwise overlap.

#### 3.3.2.2 owl:minCardinality

The cardinality constraint *owl:minCardinality* is a built-in OWL property that links a restriction class to a data value belonging to the value space of the XML Schema datatype nonNegativeInteger. A restriction containing an *owl:minCardinality* constraint describes a class of all individuals that have at least $N$ semantically distinct values (individuals or data values) for the property concerned, where $N$ is the value of the cardinality constraint. Syntactically, the cardinality constraint is represented as an RDF property element with the corresponding rdf:datatype attribute.

Our approach for the temporal version of this constraint is that it describes a class of all individuals that are connected to at least $N$ members of the *Event* class with the property concerned. The event individuals have at least one value for the property concerned.

OWL adopts the *open world assumption*, thus if a member of a class restricted with a *minCardinality* constraint has less than N distinct values for the concerned property, no inconsistency will result.

An example of an *owl:minCardinality* constraint would be:

*employs* ***min 2*** *Employee*

The temporal version of that minCardinality constraint is:

*employs* ***min 2*** *(Event* ***and*** *(employs* ***some*** *Employee))*

Actually this interpretation of the constraint does not imply that the individuals of the class 'Company' have at least 2 employees at the same time, but just that they have two employees in their existence, connected with the temporal property *employs*. This is a less strict appliance of the right constraint which would require all the company individuals to have at least 2 employees at the same time. This was the only temporal interpretation of the minCardinality constraint we were able to implement, given the current expressiveness of property restrictions and SWRL rules.

#### 3.3.2.3 owl:cardinality

The cardinality constraint *owl:cardinality* is a built-in OWL property that links a restriction class to a data value belonging to the range of the XML Schema datatype nonNegativeInteger. A restriction containing an *owl:cardinality* constraint describes a class of all individuals that have exactly $N$ semantically distinct values (individuals or data values) for the property concerned, where $N$ is the value of the cardinality constraint. Syntactically, the cardinality constraint is represented as an RDF property element with the corresponding rdf:datatype attribute.

Our approach for this constraint as temporal is that it describes a class of all individuals that are related to *Event* individuals with the property

concerned and those event individuals have exactly $N$ semantically distinct values for the property concerned. An example of an *owl:cardinality* constraint would be:

*employs* ***exactly 2*** *Employee*

It implies that an individual of the class 'Company' can be related to exactly two individuals of the class Employee with the object property *employs*. The SWRL rule that would be added to the ontology to apply the temporal version of the constraint would be:

*Event(?e0)* $\wedge$*Event(?e1)* $\wedge$*Event(?e2)* $\wedge$*Company(?x)* $\wedge$*Employee(?y0)*
$\wedge$*Employee(?y1)*$\wedge$*Employee(?y2)*$\wedge$*during(?e0, ?i0)*$\wedge$*during(?e1, ?i1)*
$\wedge$*during(?e2, ?i2)* $\wedge$*overlaps(?i0, ?i1)* $\wedge$*overlaps(?i0, ?i2)*
$\wedge$*overlaps(?i1, ?i2)* $\wedge$*employs(?e0, ?y0)* $\wedge$*employs(?e1, ?y1)*
$\wedge$*employs(?e2, ?y2)* $\wedge$*employs(?x, ?e0)*$\wedge$*employs(?x, ?e1)*
$\wedge$*employs(?x, ?e2)*$\wedge$*DifferentFrom(?y0, ?y1)*
$\wedge$*DifferentFrom(?y0, ?y2)* $\wedge$*DifferentFrom(?y1, ?y2)*
$\rightarrow$ *Nothing(?x)*

This rule would result in inconsistency if an individual of the class 'Company' was related to more than two instances of the class 'Employee', with the temporal object property *employs*, but not if it was related to only one 'Employee' individual. So, the behavior of the temporal versions of the *maxCardinality* and the cardinality constraint coincides.

### 3.3.3 Global Cardinality Constraints on Properties

#### 3.3.3.1 owl:FunctionalProperty

A functional property is a property that can have only one (unique) value y for each instance x, i.e. there cannot be two distinct values *y1* and *y2* such that the pairs *(x, y1)* and *(x, y2)* are both instances of this property. Both object properties and datatype properties can be declared as "functional". For this purpose, OWL defines the built-in class *owl:FunctionalProperty* as a special subclass of the RDF class rdf:Property.

A temporal functional property can have only one value in each time interval during which the property holds. This is attainable by adding a SWRL rule to the ontology. For the temporal property *employs* to be functional, the rule added would be:

*Event(?e1)*$\wedge$*Event(?e2)*$\wedge$*during(?e1, ?i1)* $\wedge$ *during(?e2, ?i2)*
$\wedge$*overlaps(?i1, ?i2)* $\wedge$*employs(?e1, ?y1)*$\wedge$*employs(?e2, ?y2)*
$\wedge$*employs(?x, ?e1)* $\wedge$*employs(?x, ?e2)* $\wedge$*DifferentFrom (?y1, ?y2)*
$\rightarrow$ *SameAs (?y1, ?y2)*

#### 3.3.3.2 owl:InverseFunctionalProperty

If a property is declared to be inverse-functional, then the object of a property statement uniquely determines the subject (some individual). More formally, if we state that $P$ is an *owl:InverseFunctionalProperty*, then this asserts that a value $y$ can only be the value of $P$ for a single instance $x$, i.e. there cannot be two distinct instances *x1* and *x2* such that both pairs *(x1, y)* and *(x2, y)* are instances of $P$.

When a temporal property is inverse-functional the object uniquely determines the subject for each time instant, i.e. there can be two instances *x1*, *x2* such that *(x1, y, interval1)* and *(x2, y, interval2)* are instances of $P$ as long as the *interval1* and *interval2* do not overlap. The SWRL rule to make the temporal property *worksFor* inverse-functional is:

*Event(?e1)∧Event(?e2)∧during(?e1, ?i1) ∧ during(?e2, ?i2)*
*∧overlaps(?i1, ?i2) ∧worksFor(?e1, ?y)∧worksFor(?e2, ?y)*
*∧worksFor(?x1, ?e1) ∧worksFor(?x2, ?e2)*
*∧DifferentFrom(?x1, ?x2) → SameAs (?x1, ?x2)*

### 3.3.4 Negative property assertions

A *negative property assertion* is a feature introduced by OWL 2 and it is a kind of restriction applied on individuals, not allowing them to have a specific value (individual or data value). More formally, a negative property assertion between the individual $x$, the value $y$, connected with the property $P$ asserts that there cannot be an instance of the property such as *P(x,y)*. A temporal negative property assertion restricts an individual $x$ in a way that it cannot be connected with a temporal property $P$ to a specific value $y$ during a time interval *interval1*, thus the *P(x,y,interval1)* cannot be an instance of the temporal property $P$. The SWRL rule added to the ontology would be:

*Event(?e)∧during(?e, ?i)∧overlaps(?i, interval1)*
*∧employs(?e, John)∧employs(Company1, ?e)*
*→ Nothing(Company1)*

This rule forbids the individual 'Company1' to employ 'John' during any time interval that somehow overlaps with the specified time interval 'interval1'.

### 3.3.5 Transitive Properties

When a property is defined to be transitive, this means that if a pair *(x, y)* is an instance of $P$, and the pair *(y, z)* is an instance of $P$, then we can infer that the pair *(x, z)* is also an instance of $P$.

The instances of the properties are considered to hold for a specific time interval. If a temporal property $P$ is transitive and *(x, y, interval1)* is an instance of $P$ and *(y, z, interval2)* is an instance of $P$, then we can infer that *(x, z, interval1∩interval2)* is also an instance of $P$. The SWRL expressiveness though does not allow the creation of instances of classes. Thus, the creation of such a rule was not possible.

In our implementation the transitivity between instances of a temporal property takes place *only* if those instances hold for same time intervals. The SWRL applying that effect on the temporal property *worksFor*, would be:

*Event(?e1)∧Event(?e2)∧during(?e1, ?i1)∧during(?e2, ?i2)*
*∧worksFor(?e1, ?y)∧worksFor(?e2, ?z)∧worksFor(?x, ?e1)*
*∧worksFor(?y, ?e2)∧intervalEquals(?i1, ?i2)*
*∧DifferentFrom (?y, ?z) → worksFor(?x, ?e2)*

# Chapter 4

# Implementation

## 4.1 Use Cases

Chronos is a plug-in used for representing temporal information in Protégé. It requires a vocabulary that describes temporal concepts. Our version of OWL-Time [3] distributed with Chronos provides this vocabulary along with a set of SWRL rules, developed by Batsakis [9, 10], that allow reasoning over temporal relations and temporal concepts.

When someone selects Chronos Tab for the first time, our tool checks if the active ontology is merged with the OWL-Time ontology. If it is not, a pop-up window will appear, prompting the user to merge the ontology with the OWL-Time ontology. The user may select not to merge the active ontology. In that case, Chronos will add to the ontology only the OWL entities required for the representation of temporal relationships, but will not provide any reasoning capabilities or consistency checking over the temporal concepts of the ontology.

Table 4.1 describes the use case where the user converts an object property to temporal. The user selects an object property and presses the "Convert" button[1]. Since this property will be converted to temporal, all the triples containing it will be converted too. A pop-up dialog window appears that informs the user about the triples that will be affected by the conversion. The user may specify a time interval during which those temporal triples will hold, or leave that field blank. In the later case Chronos connects each temporal triple to an (different) unknown time interval. The user may also cancel the conversion. In that case no changes will occur in the ontology.

The case of the conversion of a data property to temporal is identical to that of the object property. When the user clicks on the "Convert" button,

[1] Chonos will only allow the conversion of entities that do not belong in OWL-Time ontology, and they are not used by the N-ary Relations model to represent temporal properties (i.e. the object property *during*).

the “Affected triples” dialog window will pop-up, informing him of the triples that will also be converted to temporal. The user may either proceed to the conversion by clicking on the “Yes” button, or cancel it by clicking on the “Cancel” button. Table 4.2 describes this use case.

Chronos also enables the user to convert classes to temporal (as explained in subsection 3.2.4). When a class is selected from the “Class Hierarchy View”, the “Class” panel is displayed and the “Convert” button is enabled. When the “Convert” button is pressed, a pop-up window appears informing the user about the object and data properties that will be converted to temporal as well as about the triples that will be affected by those conversions. Similarly to the object property use case, the user may select the time interval during which all these temporal properties will hold. This use case is described in table 4.3.

The user can add a temporal restriction on a class. In order to add a constraint as an equivalent class or as a superclass, the user selects a class and presses the “Add” button in the “Classes” panel next to the “Equivalent classes” or the “Superclasses” list respectively. Then, the “Class restriction editor” window will appear, where the user can create the temporal property restriction. The user must specify:

1. the temporal restricted property,
2. the restriction filler,
3. the restriction type (and the cardinality for cardinality constraints).

By clicking on the “OK” button the restriction is added on the selected class. This use case is described in table 4.4.

Editing a temporal property constraint is similar to this of the addition. The user selects a temporal property constraint from the “Equivalent classes” or the “Superclasses” list and clicks on the “Edit” button. Then the “Class restriction editor” window appears and the user creates the restriction exactly in the same way to the addition of a temporal property constraint. This use case is described in table 4.5.

The added restrictions can easily be removed by clicking on the “Remove” button next to the “Equivalent classes” or the “Superclasses” list in the “Class” panel. Table 4.6 describes this use case.

One can also create an individual that is in advance related with a temporal object or data property to another individual. The user selects a class from the “Class Hierarchy View” and then presses the “New Individual” button from the “Class” panel. Then the “Individual Wizard” window will appear, that is a four-step wizard which will guide the user through the creation of the temporal triple. In these steps the user:

1. Sets the name of the new individual.

2. Selects the temporal object or data property that will connect the new individual (subject) to an already existing individual (object).

3. Sets the time interval. This interval may have specified bounds or non-specified bounds (qualitative).

4. Reviews the selections made and confirms the creation of the temporal triple.

The use case of the creation of a temporal triple is described in table 4.7.

An object or data property assertion can be added to an individual even after its creation. The user selects an individual and he presses the "Add" button next to the "Object property assertions" or the "Data property assertions" list. The "Individual Wizard" window appears, guiding the user just as in the creation of a temporal triple. This use case is described in table 4.8.

The interval during which a temporal property assertion holds can also be edited. The user selects the concerned individual from the "Individual by type" view and the "Individual" panel is displayed, showing the object and data property assertions of the selected individual. The user selects a temporal object or data property assertion and presses the "Edit" button that is next to the "Object property assertions" or the "Data property assertions" list respectively. Then, the "Individual Wizard" pops-up, displaying the step where the user sets the time interval. Finally, the interval is created and it is connected to the temporal triple. The time interval that has been replaced is not removed from the ontology, as it may be connected to other temporal triples. This use case is described in table 4.9.

Chronos facilitates the creation and the editing of temporal triples as well as their removal. The user selects the temporal object or data property assertion and presses the "Remove" button next to the corresponding list. This use case is described in table 4.10.

The user may also add, edit or remove temporal *negative* object or data property assertions on individuals. This can be done in the same way to temporal object and data property assertions. The only difference is that the user must press the "Add", "Edit" or "Remove" button next to the "Negative object property assertions" and "Negative data property assertions" in the "Individual" panel. The use cases concerning these operations are described in tables 4.11, 4.12 and 4.13, respectively.

The selection of the time interval is a basic step in almost every use case mentioned. The time interval may or may not have specific bounds. A time interval may have (a) unknown bounds, (b) specific start or end (but not both) or (c) it can be qualitative, meaning that it is connected with a qualitative property (Allens relations [14] such as 'after', 'before' etc.) to another time interval. Table 4.14 describes the steps of the creation of time

intervals, either during the conversion of an entity to temporal, or during the creation of property assertions between individuals.

| **Use Case 1** | **Convert an object property to temporal** | |
|---|---|---|
| **Goal In Context** | The user wants to convert a static object property to temporal | |
| **Scope & Level** | System, Main functionality | |
| **Preconditions** | There are object properties in the ontology<br>Chronos Tab is selected | |
| **Success End Condition** | The object property and all the triples, where it appears in as a predicate, are temporal | |
| **Failed End Condition** | The object property is static. No changes have occurred in the ontology | |
| **Primary Actors** | User | |
| **Trigger** | The user selects an object property and presses the "Convert" button from the "Object Property" panel | |
| **Main Path** | **Step** | **Action** |
| | 1 | The "Affected Triples" dialog window is displayed |
| | 2 | The user specifies the interval during which the temporal triples hold |
| | 3 | The user presses the "Yes" button |
| | 4 | The object property, the triples and the restrictions containing it are converted to temporal |
| | 5 | The system displays the "Object Property" panel for the selected object property |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | 3a | *The user presses the "No" button* |
| | 4a | The selected object property remains static |
| | 5a | The "Object Property" panel is displayed |
| **Sub-Variation** | **Step** | **Variation** |
| | 2.1 | *The user presses the "Add" button in the "Affected Triples" dialog window* |
| | 2.1.1 | The "Interval Creator" window is displayed |
| | 2.1.2 | The user selects the interval during which the temporal property holds |
| | 2.1.3a.1 | The user presses the "OK" button |
| | 2.1.3a.2 | The "Affected Triples" dialog is displayed, with the "Selected interval" field completed |
| | 2.1.3b.1 | The user presses the "Cancel" button |
| | 2.1.3b.1 | The "Affected Triples" dialog is displayed, with the "Selected interval" field empty |
| | 2.2 | *The user presses the "Clear" button in the "Affected Triples" dialog window* |
| | 2.2.1 | The "Selected interval" field is cleared |

Table 4.1: Use Case 1. Convert an object property to temporal

| **Use Case 2** | **Convert a data property to temporal** | |
|---|---|---|
| **Goal In Context** | The user wants to convert a static data property to temporal | |
| **Scope & Level** | System, Main functionality | |
| **Preconditions** | There are data properties in the ontology<br>Chronos Tab is selected | |
| **Success End Condition** | The data property and all the triples, where it appears in as a predicate, are temporal | |
| **Failed End Condition** | The data property is static. No changes have occurred in the ontology | |
| **Primary Actors** | User | |
| **Trigger** | The user selects a data property and presses the "Convert" button from the "Data Property" panel | |
| **Main Path** | **Step** | **Action** |
| | 1 | The "Affected Triples" dialog window is displayed |
| | 2 | The user specifies the interval during which the temporal triples hold |
| | 3 | The user presses the "Yes" button |
| | 4 | The data property, the triples and the restrictions containing it are converted to temporal |
| | 5 | The system displays the "Data Property" panel for the selected object property |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | 3a | *The user presses the "No" button* |
| | 4a | The selected data property remains static |
| | 5a | The "Data Property" panel is displayed |
| **Sub-Variation** | **Step** | **Variation** |
| | 2.1 | *The user presses the "Add" button in the "Affected Triples" dialog window* |
| | 2.1.1 | The "Interval Creator" window is displayed |
| | 2.1.2 | The user selects the interval during which the temporal property holds |
| | 2.1.3a.1 | The user presses the "OK" button |
| | 2.1.3a.2 | The "Affected Triples" dialog is displayed, with the "Selected interval" field completed |
| | 2.1.3b.1 | The user presses the "Cancel" button |
| | 2.1.3b.1 | The "Affected Triples" dialog is displayed, with the "Selected interval" field empty |
| | 2.2 | *The user presses the "Clear" button in the "Affected Triples" dialog window* |
| | 2.2.1 | The "Selected interval" field is cleared |

Table 4.2: Use Case 2. Convert a Data property to temporal

| **Use Case 3** | **Convert a class to temporal** | |
|---|---|---|
| **Goal In Context** | The user wants to convert a static class to temporal | |
| **Scope & Level** | System, Main functionality | |
| **Preconditions** | There are classes in the ontology<br>Chronos Tab is selected | |
| **Success End Condition** | All the object and data properties that are related to the selected class, as well as the properties that connect instances of the selected class, are temporal. | |
| **Failed End Condition** | The properties related to this class are static. No changes have occurred in the ontology | |
| **Primary Actors** | User | |
| **Trigger** | The user selects a class and presses the "Convert" button from the "Class" panel | |
| **Main Path** | **Step** | **Action** |
| | 1 | The "Affected Entities" dialog window is displayed |
| | 2 | The user specifies the interval during which the temporal properties and triples will hold |
| | 3 | The user presses the "Yes" button |
| | 4 | The object and data properties and the restrictions concerning this class, as well as its individuals are converted to temporal |
| | 5 | The system displays the "Class" panel for the selected temporal class |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | 3a | *The user presses the "No" button* |
| | 4a | The selected class remains static |
| | 5a | The "Class" panel is displayed |
| **Sub-Variation** | **Step** | **Variation** |
| | 2.1 | *The user presses the "Add" button in the "Affected Triples" dialog window* |
| | 2.1.1 | The "Interval Creator" window is displayed |
| | 2.1.2 | The user selects the interval during which the temporal property holds |
| | 2.1.3a.1 | The user presses the "OK" button |
| | 2.1.3a.2 | The "Affected Triples" dialog is displayed, with the "Selected interval" field completed |
| | 2.1.3b.1 | The user presses the "Cancel" button |
| | 2.1.3b.1 | The "Affected Triples" dialog is displayed, with the "Selected interval" field empty |
| | 2.2 | *The user presses the "Clear" button in the "Affected Triples" dialog window* |
| | 2.2.1 | The "Selected interval" field is cleared |

Table 4.3: Use Case 3. Convert a class to temporal

| **Use Case 4** | **Add a temporal property restriction** | |
|---|---|---|
| **Goal In Context** | The user wants to add a temporal property constraint as an equivalent or a superclass to the selected class | |
| **Scope & Level** | System, Main functionality | |
| **Preconditions** | There is at least one temporal object or data property in the ontology. Chronos Tab is selected | |
| **Success End Condition** | The temporal property constraint has been added as an equivalent or a superclass to the selected class | |
| **Failed End Condition** | No changes have occurred in the ontology | |
| **Primary Actors** | User | |
| **Trigger** | The user selects a class and presses the "Add" button next to the "Equivalent class" or the "Superclass" list in the "Class" panel | |
| **Main Path** | **Step** | **Action** |
| | 1 | The "Class restriction editor" window pops-up |
| | 2 | The user selects the property to be restricted |
| | 3 | The user selects the restriction filler from the "Restriction filler" list |
| | 4 | The user specifies the restriction type |
| | 5 | The user presses the "OK" button and the creation is complete |
| | 6 | The system displays the "Class" panel and the created restriction appears in the "Equivalent class" or the "Superclass" list |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | 2a | *The user presses the "Cancel" button* |
| | 3a | The creation is canceled |
| | 4a | The "Class" panel is displayed |
| **Sub-Variation** | **Step** | **Variation** |
| | 2a | *The user selects the "Object property" radio button* |
| | 2a.1 | The "Restricted property" list is filled with the available temporal object properties and the "Restriction filler" list is filled with the available classes |
| | 2b | *The user selects the "Data property" radio button* |
| | 2b.1 | The "Restricted property" list is filled with the available temporal data properties and the "Restriction filler" list is filled with the available data types |
| | 4a | *The user selects a cardinality constraint* |
| | 4a.1 | The cardinality selection field is enabled |
| | 4a.2 | The user sets the desired cardinality |

Table 4.4: Use Case 4. Add a temporal property restriction

| **Use Case 5** | **Edit a temporal property restriction** | |
|---|---|---|
| **Goal In Context** | The user wants to edit a temporal property constraint that appears as an equivalent or a superclass to the selected class | |
| **Scope & Level** | System, Main functionality | |
| **Preconditions** | There is at least one temporal property restriction. Chronos Tab is selected | |
| **Success End Condition** | The temporal property constraint has been modified | |
| **Failed End Condition** | The temporal property constraint has not been modified | |
| **Primary Actors** | User | |
| **Trigger** | The user selects a temporal property constraint from the "Equivalent classes" or the "Superclasses" list and presses the "Edit" button that is next to this list in the "Class" panel | |
| **Main Path** | **Step** | **Action** |
| | 1 | The "Class restriction editor" window pops-up. The "Restricted property", "Restriction filler" and "Restriction type" selections are set to those of the restriction to be edited |
| | 2 | The user selects the property to be restricted |
| | 3 | The user selects the restriction filler from the "Restriction filler" list |
| | 4 | The user specifies the restriction type |
| | 5 | The user presses the "OK" button and the creation is complete |
| | 6 | The system displays the "Class" panel and the created restriction appears in the "Equivalent class" or the "Superclass" list |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | 2a | *The user presses the "Cancel" button* |
| | 3a | The creation is canceled |
| | 4a | The "Class" panel is displayed |
| **Sub-Variation** | **Step** | **Variation** |
| | 2a | *The user selects the "Object property" radio button* |
| | 2a.1 | The "Restricted property" list is filled with the available temporal object properties and the "Restriction filler" list is filled with the available classes |
| | 2b | *The user selects the "Data property" radio button* |
| | 2b.1 | The "Restricted property" list is filled with the available temporal data properties and the "Restriction filler" list is filled with the available data types |
| | 4a | *The user selects a cardinality constraint* |
| | 4a.1 | The cardinality selection field is enabled |
| | 4a.2 | The user sets the desired cardinality |

Table 4.5: Use Case 5. Edit a temporal property restriction

| **Use Case 6** | **Remove a class description** | |
|---|---|---|
| **Goal In Context** | The user wants to remove a class description that appears as an equivalent or a superclass to the selected class | |
| **Scope & Level** | System, Main functionality | |
| **Preconditions** | The selected class has at least one class description as an equivalent class or superclass.<br>Chronos Tab is selected | |
| **Success End Condition** | The class description has been removed | |
| **Failed End Condition** | The class description has not been modified | |
| **Primary Actors** | User | |
| **Trigger** | The user selects a class description from the "Equivalent classes" or the "Superclasses" list and presses the "Remove" button that is next to this list in the "Class" panel | |
| **Main Path** | **Step** | **Action** |
| | 1 | The "Removal Confirmation" dialog window pops-up |
| | 2 | The user presses the "Yes" button |
| | 3 | The class description has been removed |
| | 4 | The system displays the "Class" panel |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | 2a | *The user presses the "No" button* |
| | 3a | The removal is canceled |
| | 4a | The "Class" panel is displayed |

Table 4.6: Use Case 6. Remove a class description

| **Use Case 7** | **Create a temporal individual** | |
|---|---|---|
| **Goal In Context** | The user wants to create an individual that is connected with a temporal property to an already existing individual or data value | |
| **Scope & Level** | System, Main functionality | |
| **Preconditions** | There is at least one temporal object or data property in the ontology. Chronos Tab is selected | |
| **Success End Condition** | The temporal triple has been created and added to the ontology | |
| **Failed End Condition** | No changes have occurred in the ontology | |
| **Primary Actors** | User | |
| **Trigger** | The user selects a class and presses the "New Individual" button from the "Class" panel | |
| **Main Path** | **Step** | **Action** |
| | 1 | The "Individual Wizard" window pops-up |
| | 2 | The user sets the name of the new individual |
| | 3 | The user selects the property that will connect the subject individual to the object individual. |
| | 4 | The user specifies the interval during which the assertion will hold |
| | 5 | The user presses the "Finish" button and the creation is complete |
| | 6 | The system displays the "Class" panel |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | any | *The user presses the "Cancel" button* |
| | .1 | The creation is canceled |
| | .2 | The "Class" panel is displayed |
| **Alternative Path (A2)** | **Step** | **Branching Action** |
| | any | *The user presses the "Previous" button* |
| | .1 | The system returns to the previous step of the creation |
| **Sub-Variation** | **Step** | **Variation** |
| | 4.1a | *The user selects the "Create a new interval" radio button* |
| | 4.1a.1a | The user selects the "Specific interval" radio button. and specifies the start date AND the end date of the interval |
| | 4.1a.1b | The user selects the "Non-specific" interval radio button and specifies the start date OR the end date of the interval. He may select a qualitative property which will relate the new interval to an already existing one. |
| | 4.1b | *The user selects the "Select an existing interval" radio button and selects one of the intervals of the list* |
| | 4.2 | The user presses the "Add interval" button and the new interval appears in the "Selected interval" text field |
| | 4.3 | The user clicks on the "Next" button |

Table 4.7: Use Case 7. Create a temporal individual

| **Use Case 8** | **Add a temporal object or data property assertion** | |
|---|---|---|
| **Goal In Context** | The user wants to add a temporal property assertion to the selected individual | |
| **Scope & Level** | System, Main functionality | |
| **Preconditions** | There is at least one temporal object or data property in the ontology. Chronos Tab is selected | |
| **Success End Condition** | The created property assertion has been added to the selected individual | |
| **Failed End Condition** | No changes have occurred in the ontology | |
| **Primary Actors** | User | |
| **Trigger** | The user selects an individual and presses the "Add" button, next to the "Object property assertions" or the "Data property assertions" in the "Individual" panel | |
| **Main Path** | **Step** | **Action** |
| | 1 | The "Individual Wizard" window pops-up |
| | 2 | The user selects one of the available properties |
| | 3 | The user specifies the interval during which the assertion will hold |
| | 4 | The user presses the "Finish" button and the creation is complete |
| | 5 | The system displays the "Class" panel |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | any | *The user presses the "Cancel" button* |
| | .1 | The addition is canceled |
| | .2 | The "Class" panel is displayed |
| **Alternative Path (A2)** | **Step** | **Branching Action** |
| | any | *The user presses the "Previous" button* |
| | .1 | The system returns to the previous step of the creation |
| **Sub-Variation** | **Step** | **Variation** |
| | 4.1a | *The user selects the "Create a new interval" radio button* |
| | 4.1a.1a | The user selects the "Specific interval" radio button. and specifies the start date AND the end date of the interval |
| | 4.1a.1b | The user selects the "Non-specific" interval radio button and specifies the start date OR the end date of the interval. He may select a qualitative property which will relate the new interval to an already existing one. |
| | 4.1b | *The user selects the "Select an existing interval" radio button and selects one of the intervals of the list* |
| | 4.2 | The user presses the "Add interval" button and the new interval appears in the "Selected interval" text field |
| | 4.3 | The user clicks on the "Next" button |

Table 4.8: Use Case 8. Add a temporal property assertion

| **Use Case 9** | **Edit the interval of a temporal object or data property assertion** | |
|---|---|---|
| **Goal In Context** | The user wants to edit the interval during which a temporal triple holds | |
| **Scope & Level** | System, Main functionality | |
| **Preconditions** | There is at least one individual connected with a temporal object or data property. Chronos Tab is selected | |
| **Success End Condition** | The existing time interval has been replaced with the one specified by the user | |
| **Failed End Condition** | The existing time interval has not been replaced | |
| **Primary Actors** | User | |
| **Trigger** | The user selects a temporal object or data property assertion and presses the "Edit" button, next to the "Object property assertions" or the "Data property assertions" in the "Individual" panel | |
| **Main Path** | **Step** | **Action** |
| | 1 | The "Individual Wizard" window pops-up |
| | 2 | The user specifies the interval during which the assertion will hold |
| | 3 | The user presses the "Finish" button and the creation is complete |
| | 4 | The system displays the "Class" panel |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | any | *The user presses the "Cancel" button* |
| | .1 | The replacement of the interval is canceled |
| | .2 | The "Class" panel is displayed |
| **Alternative Path (A2)** | **Step** | **Branching Action** |
| | any | *The user presses the "Previous" button* |
| | .1 | The system returns to the previous step of the creation |
| **Sub-Variation** | **Step** | **Variation** |
| | 2.1a | *The user selects the "Create a new interval" radio button* |
| | 2.1a.1a | The user selects the "Specific interval" radio button. and specifies the start date AND the end date of the interval |
| | 2.1a.1b | The user selects the "Non-specific" interval radio button and specifies the start date OR the end date of the interval. He may select a qualitative property which will relate the new interval to an already existing one. |
| | 2.1b | *The user selects the "Select an existing interval" radio button and selects one of the intervals of the list* |
| | 2.2 | The user presses the "Add interval" button and the new interval appears in the "Selected interval" text field |
| | 2.3 | The user clicks on the "Next" button |

Table 4.9: Use Case 9. Edit the interval of a temporal property assertion

| **Use Case 10** | **Remove an object or data property assertion** | |
|---|---|---|
| **Goal In Context** | The user wants to remove an object property assertion | |
| **Scope & Level** | System, Main functionality | |
| **Preconditions** | The selected individual participates in at least one triple.<br>Chronos Tab is selected | |
| **Success End Condition** | The property assertion has been removed | |
| **Failed End Condition** | The property assertion has not been removed | |
| **Primary Actors** | User | |
| **Trigger** | The user selects a property assertion from the "Object property assertions" or the "Data property assertions" list and presses the "Remove" button that is next to this list in the "Individual" panel | |
| **Main Path** | **Step** | **Action** |
| | 1 | The "Removal Confirmation" dialog window pops-up |
| | 2 | The user presses the "Yes" button |
| | 3 | The class description has been removed |
| | 4 | The system displays the "Individual" panel |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | 2a | *The user presses the "No" button* |
| | 3a | The removal is canceled |
| | 4a | The "Individual" panel is displayed |

Table 4.10: Use Case 10. Remove a property assertion

| **Use Case 11** | **Add a temporal negative object or data property assertion** | |
|---|---|---|
| **Goal In Context** | The user wants to add a temporal negative property assertion to the selected individual | |
| **Scope & Level** | System, Main functionality | |
| **Preconditions** | There is at least one temporal object or data property in the ontology. Chronos Tab is selected | |
| **Success End Condition** | The created negative property assertion has been added to the selected individual | |
| **Failed End Condition** | No changes have occurred in the ontology | |
| **Primary Actors** | User | |
| **Trigger** | The user selects an individual and presses the "Add" button, next to the "Negative object property assertions" or the "Negative data property assertions" in the "Individual" panel | |
| **Main Path** | **Step** | **Action** |
| | 1 | The "Individual Wizard" window pops-up |
| | 2 | The user selects one of the available properties |
| | 3 | The user specifies the interval during which the negative assertion will hold |
| | 4 | The user presses the "Finish" button and the creation is complete |
| | 5 | The system displays the "Class" panel |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | any | *The user presses the "Cancel" button* |
| | .1 | The addition is canceled |
| | .2 | The "Class" panel is displayed |
| **Alternative Path (A2)** | **Step** | **Branching Action** |
| | any | *The user presses the "Previous" button* |
| | .1 | The system returns to the previous step of the creation |
| **Sub-Variation** | **Step** | **Variation** |
| | 4.1a | *The user selects the "Create a new interval" radio button* |
| | 4.1a.1a | The user selects the "Specific interval" radio button. and specifies the start date AND the end date of the interval |
| | 4.1a.1b | The user selects the "Non-specific" interval radio button and specifies the start date OR the end date of the interval. He may select a qualitative property which will relate the new interval to an already existing one. |
| | 4.1b | *The user selects the "Select an existing interval" radio button and selects one of the intervals of the list* |
| | 4.2 | The user presses the "Add interval" button and the new interval appears in the "Selected interval" text field |
| | 4.3 | The user clicks on the "Next" button |

Table 4.11: Use Case 11. Add a temporal negative property assertion

| **Use Case 12** | **Edit the interval of a temporal negative property assertion** | |
|---|---|---|
| **Goal In Context** | The user wants to edit the interval of a temporal negative property assertion to the selected individual | |
| **Scope & Level** | System, Main functionality | |
| **Preconditions** | There is at least one temporal negative property assertion in the ontology. Chronos Tab is selected | |
| **Success End Condition** | The interval of the negative property assertion has been replaced | |
| **Failed End Condition** | The existing time interval has not been replaced | |
| **Primary Actors** | User | |
| **Trigger** | The user selects an individual and presses the "Edit" button, next to the "Negative object property assertions" or the "Negative data property assertions" in the "Individual" panel | |
| **Main Path** | **Step** | **Action** |
| | 1 | The "Individual Wizard" window pops-up |
| | 2 | The user specifies the interval during which the negative assertion will hold |
| | 3 | The user presses the "Finish" button and the creation is complete |
| | 4 | The system displays the "Class" panel |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | any | *The user presses the "Cancel" button* |
| | .1 | The addition is canceled |
| | .2 | The "Class" panel is displayed |
| **Alternative Path (A2)** | **Step** | **Branching Action** |
| | any | *The user presses the "Previous" button* |
| | .1 | The system returns to the previous step of the creation |
| **Sub-Variation** | **Step** | **Variation** |
| | 4.1a | *The user selects the "Create a new interval" radio button* |
| | 4.1a.1a | The user selects the "Specific interval" radio button. and specifies the start date AND the end date of the interval |
| | 4.1a.1b | The user selects the "Non-specific" interval radio button and specifies the start date OR the end date of the interval. He may select a qualitative property which will relate the new interval to an already existing one. |
| | 4.1b | *The user selects the "Select an existing interval" radio button and selects one of the intervals of the list* |
| | 4.2 | The user presses the "Add interval" button and the new interval appears in the "Selected interval" text field |
| | 4.3 | The user clicks on the "Next" button |

Table 4.12: Use Case 12. Edit the interval of a temporal negative property assertion

| | | |
|---|---|---|
| **Use Case 13** | **Remove a negative object or data property assertion** | |
| **Goal In Context** | The user wants to remove a negative object property assertion | |
| **Scope & Level** | System, Main functionality | |
| **Preconditions** | The selected individual participates in at least one negative property assertion.<br>Chronos Tab is selected | |
| **Success End Condition** | The negative property assertion has been removed | |
| **Failed End Condition** | The negative property assertion has not been removed | |
| **Primary Actors** | User | |
| **Trigger** | The user selects a negative property assertion from the "Negative object property assertions" or the "Negative data property assertions" list and presses the "Remove" button that is next to this list in the "Individual" panel | |
| **Main Path** | **Step** | **Action** |
| | 1 | The "Removal Confirmation" dialog window pops-up |
| | 2 | The user presses the "Yes" button |
| | 3 | The class description has been removed |
| | 4 | The system displays the "Individual" panel |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | 2a | *The user presses the "No" button* |
| | 3a | The removal is canceled |
| | 4a | The "Individual" panel is displayed |

Table 4.13: Use Case 13. Remove a negative property assertion

| **Use Case 14** | **Create a time interval** | |
|---|---|---|
| **Goal In Context** | The user wants to create the time interval during which some temporal properties hold. | |
| **Scope & Level** | System, Sub functionality | |
| **Preconditions** | – – | |
| **Success End Condition** | All the required information to create the time interval have been inserted | |
| **Failed End Condition** | No adequate information | |
| **Primary Actors** | User | |
| **Trigger** | "Add" button from in the "Affected triples" window (whilst object/data property conversion, class conversion)<br>"Step 3" of the "Individual Wizard" window | |
| **Main Path** | **Step** | **Action** |
| | 1 | The user selects the "Create a new interval" radio button |
| | 2 | The user selects the "Specific interval" radio button |
| | 3 | The user specifies the start AND end date |
| | 4 | The user presses the "Add interval" button |
| | 5 | The "Selected interval" field displays the selected interval |
| | 6 | The "OK" or the "Next" button is enabled |
| | 7 | The user presses the "OK" or the "Next" button |
| **Alternative Path (A1)** | **Step** | **Branching Action** |
| | 1a | *The user selects the "Select an existing interval" radio button* |
| | 2a | A list containing all the existing intervals appears |
| | 3a | The user selects an interval from the list |
| **Alternative Path (A2)** | **Step** | **Branching Action** |
| | 2b | *The user selects the "Non-specific interval" radio button* |
| | 3b | The user specifies the start OR end date, or none |
| **Sub-Variation** | **Step** | **Variation** |
| | 3b.1 | *The user presses the "Qualitative relation" check box* |
| | 3b.2 | The "Qualitative properties" combo box is enabled and the Existing intervals list is filled |
| | 3b.3 | The user selects a qualitative property |
| | 3b.4 | The user selects an existing interval |

Table 4.14: Use Case 14. Create a time interval

## 4.2 Activity Diagrams

In this section we illustrate the activity diagrams of the use cases presented in 4.1. Each activity diagram corresponds to the use case with the same number. They present the stepwise activities and actions performed by the user of Chronos.

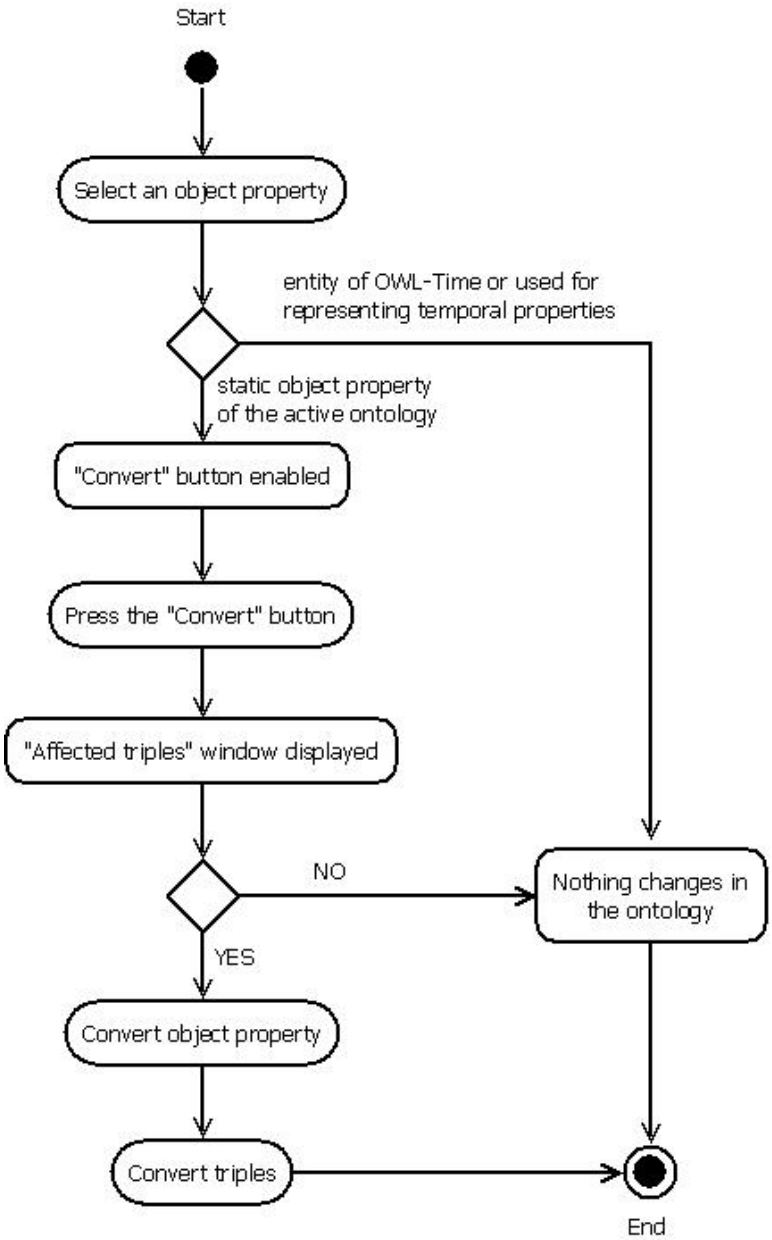


Figure 4.1: Activity Diagram 1. Convert an object property to temporal.

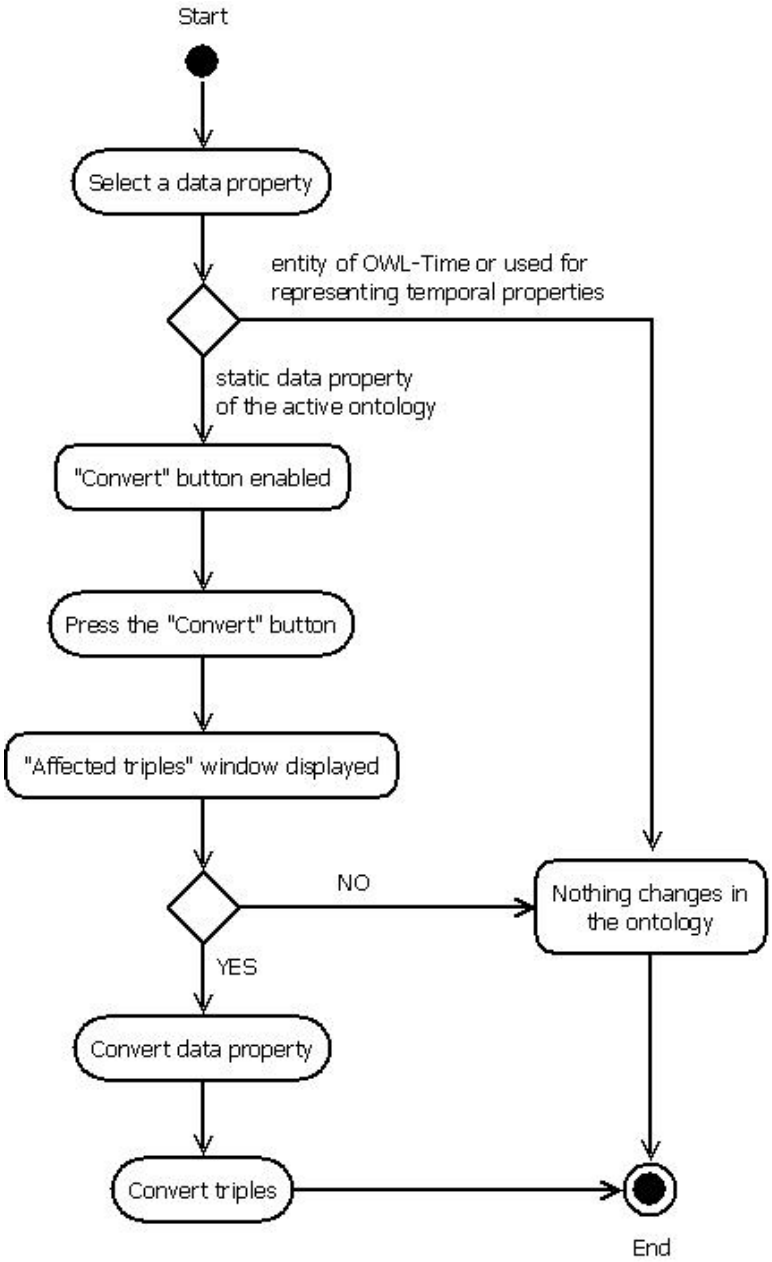


Figure 4.2: Activity Diagram 2. Convert a data property to temporal.

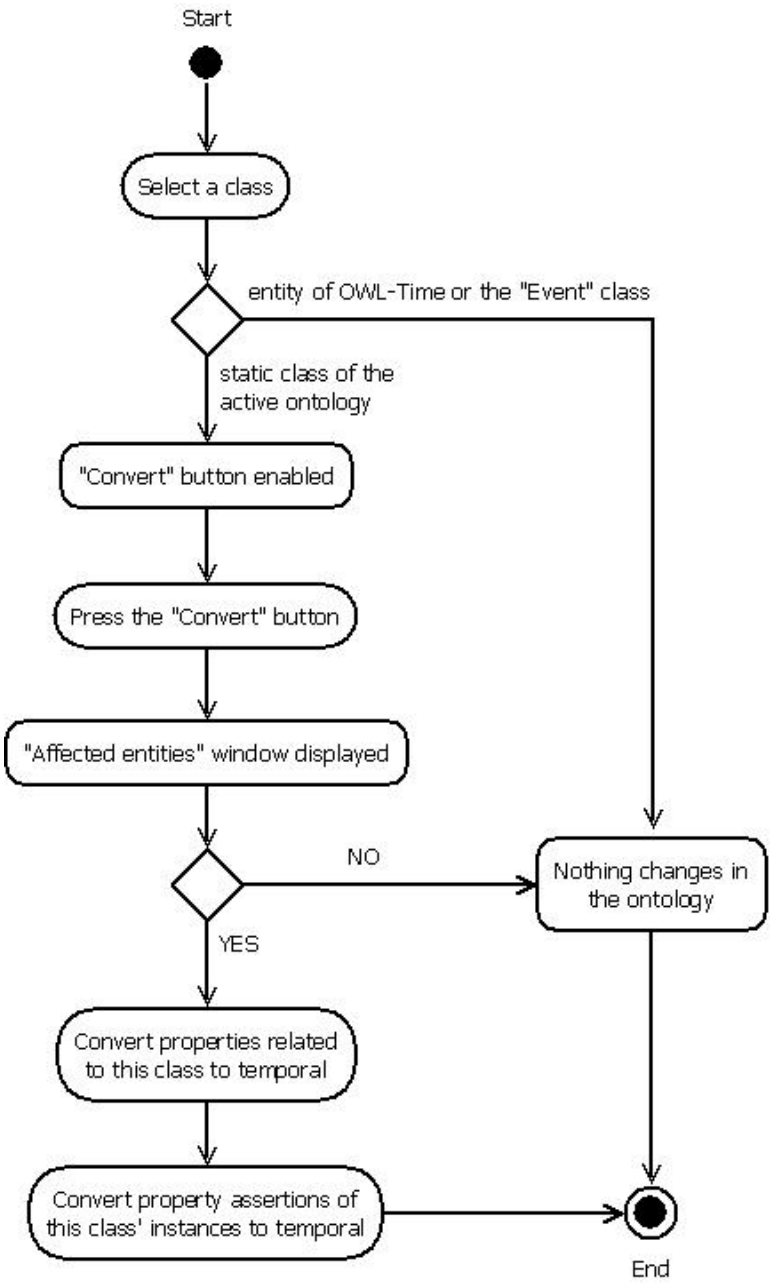


Figure 4.3: Activity Diagram 3. Convert a class to temporal.

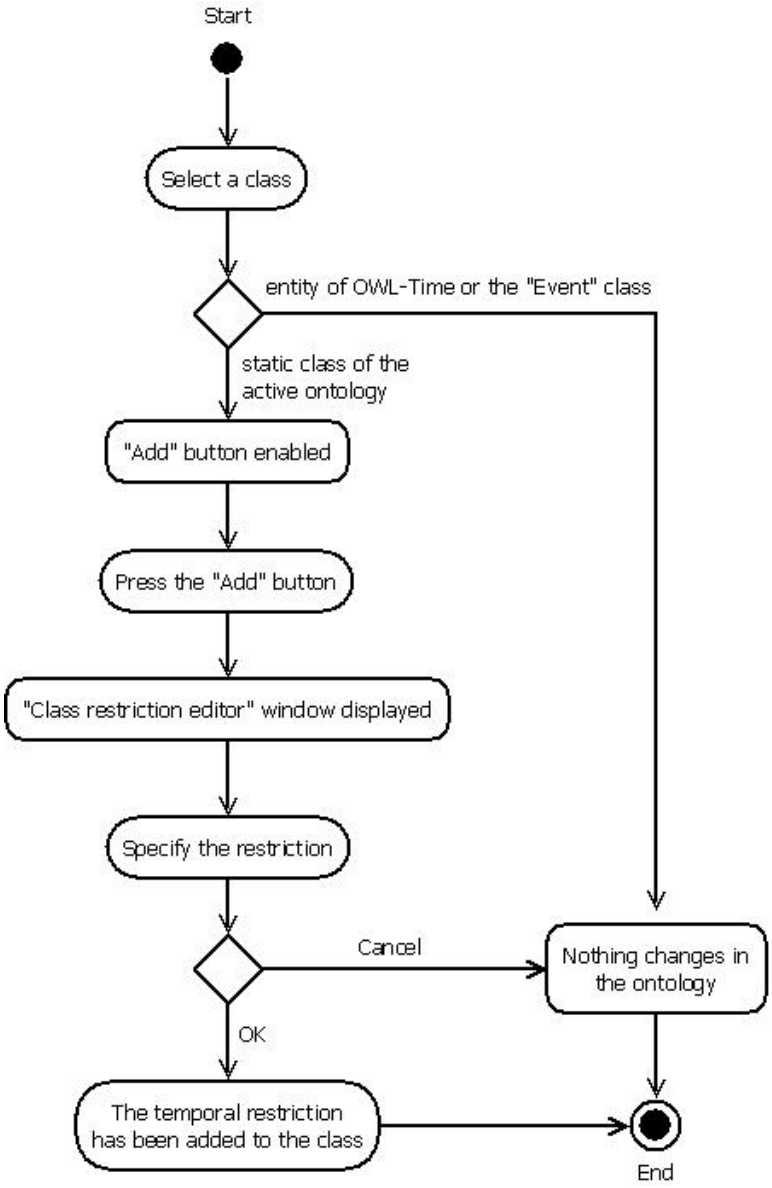


Figure 4.4: Activity Diagram 4. Add a temporal property restriction.

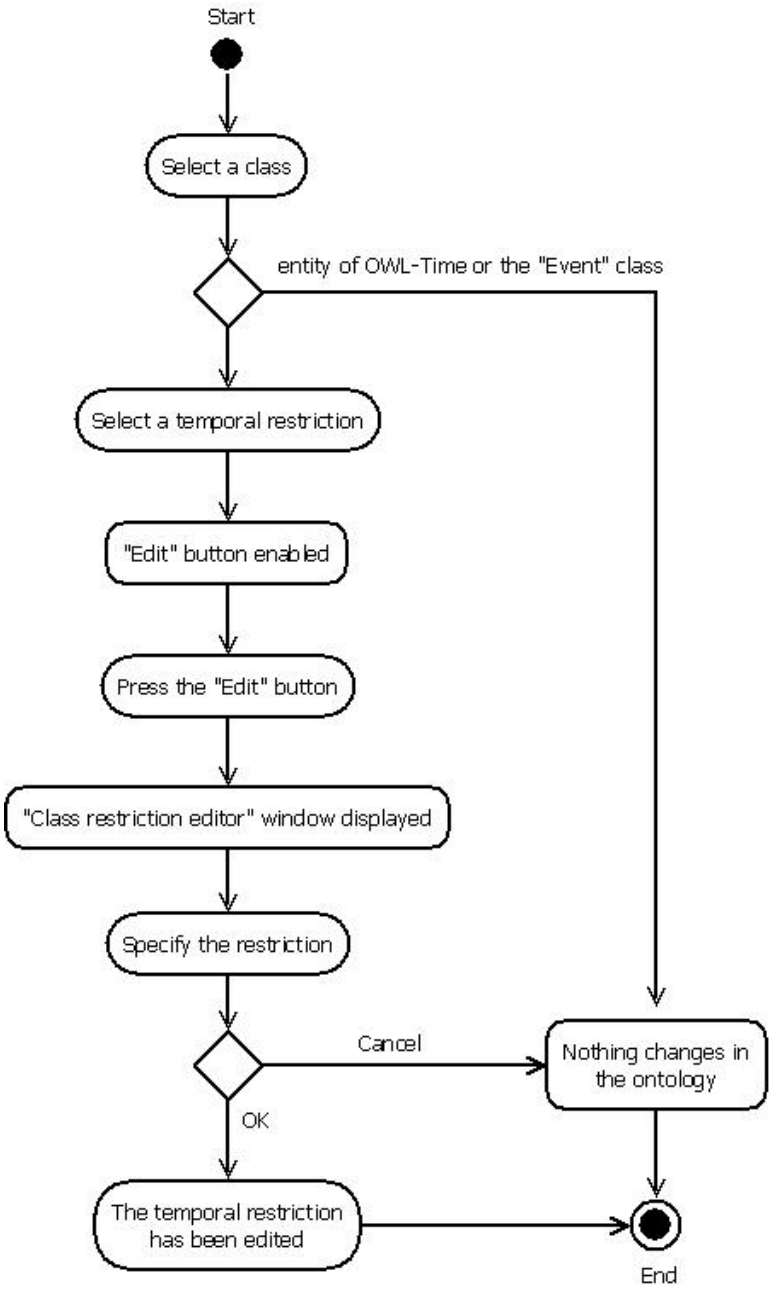


Figure 4.5: Activity Diagram 5. Edit a temporal property restriction.

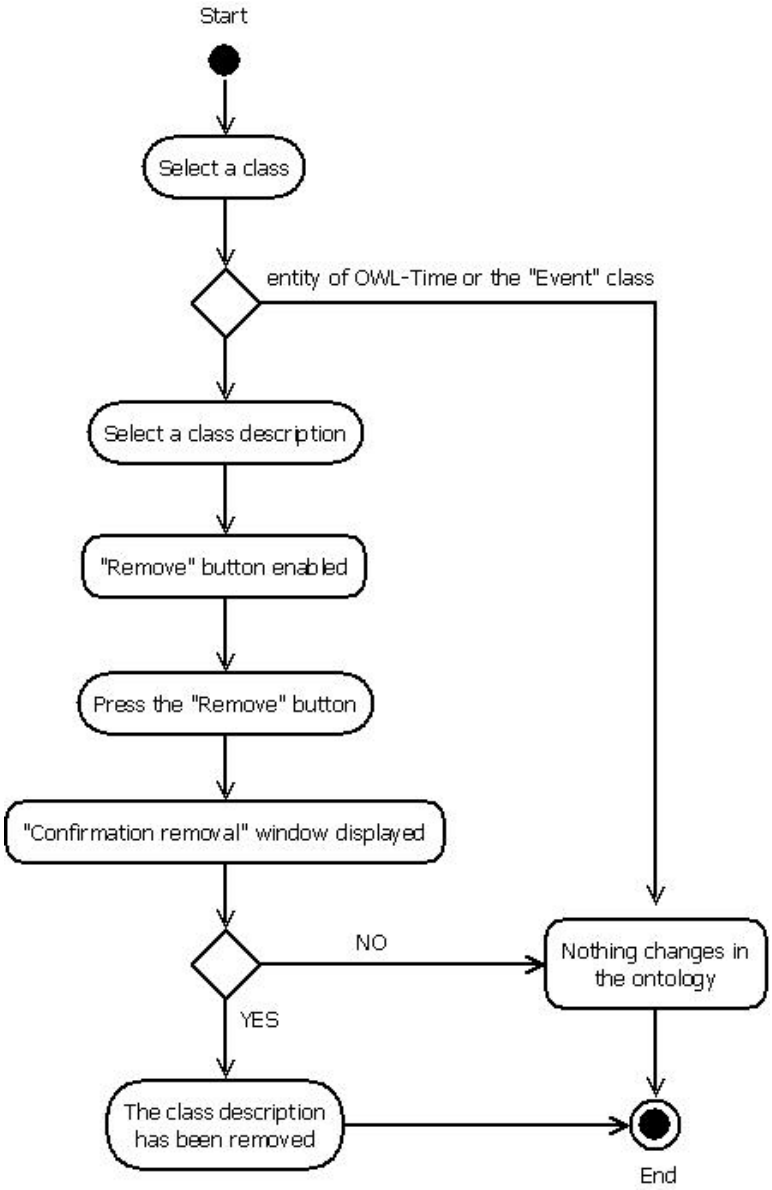


Figure 4.6: Activity Diagram 6. Remove a class description.

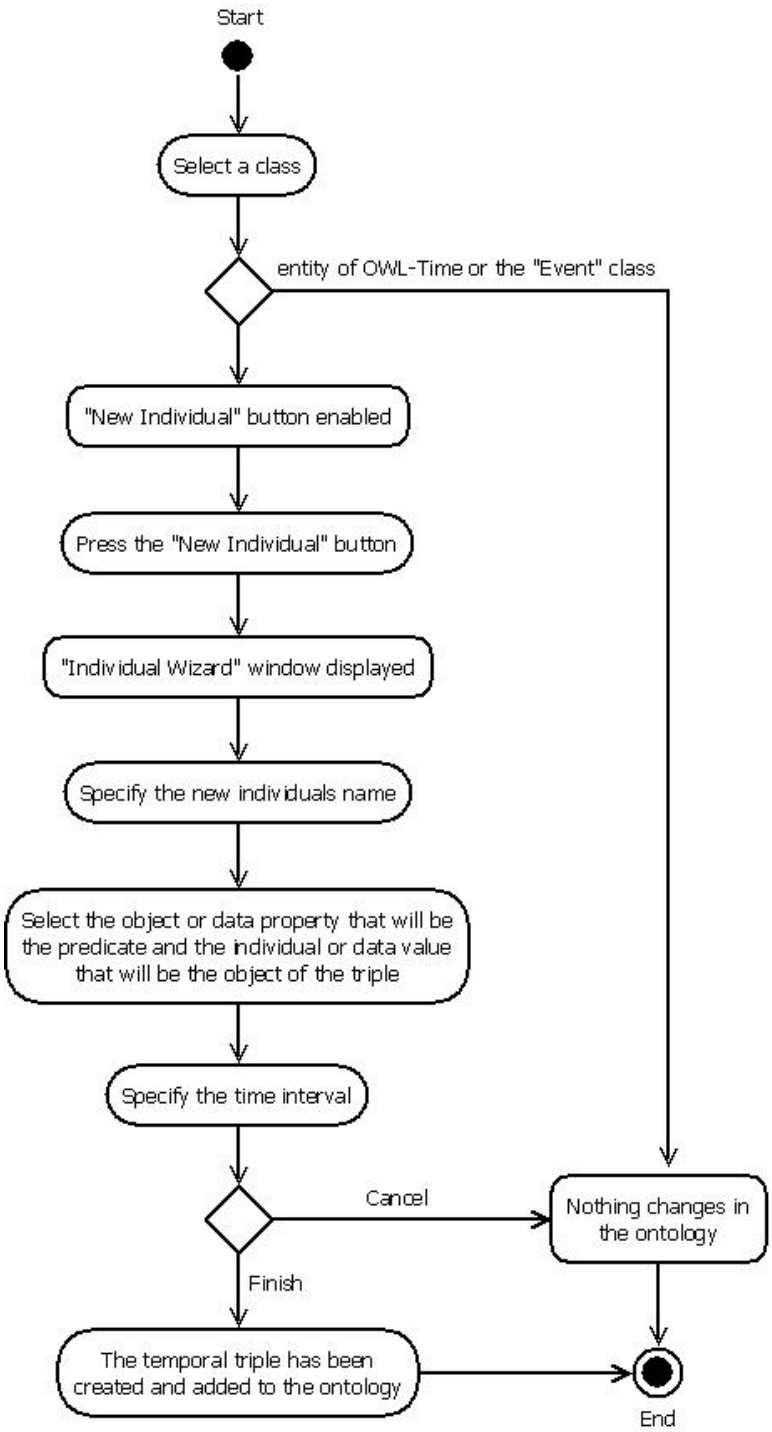


Figure 4.7: Activity Diagram 7. Create a temporal individual.

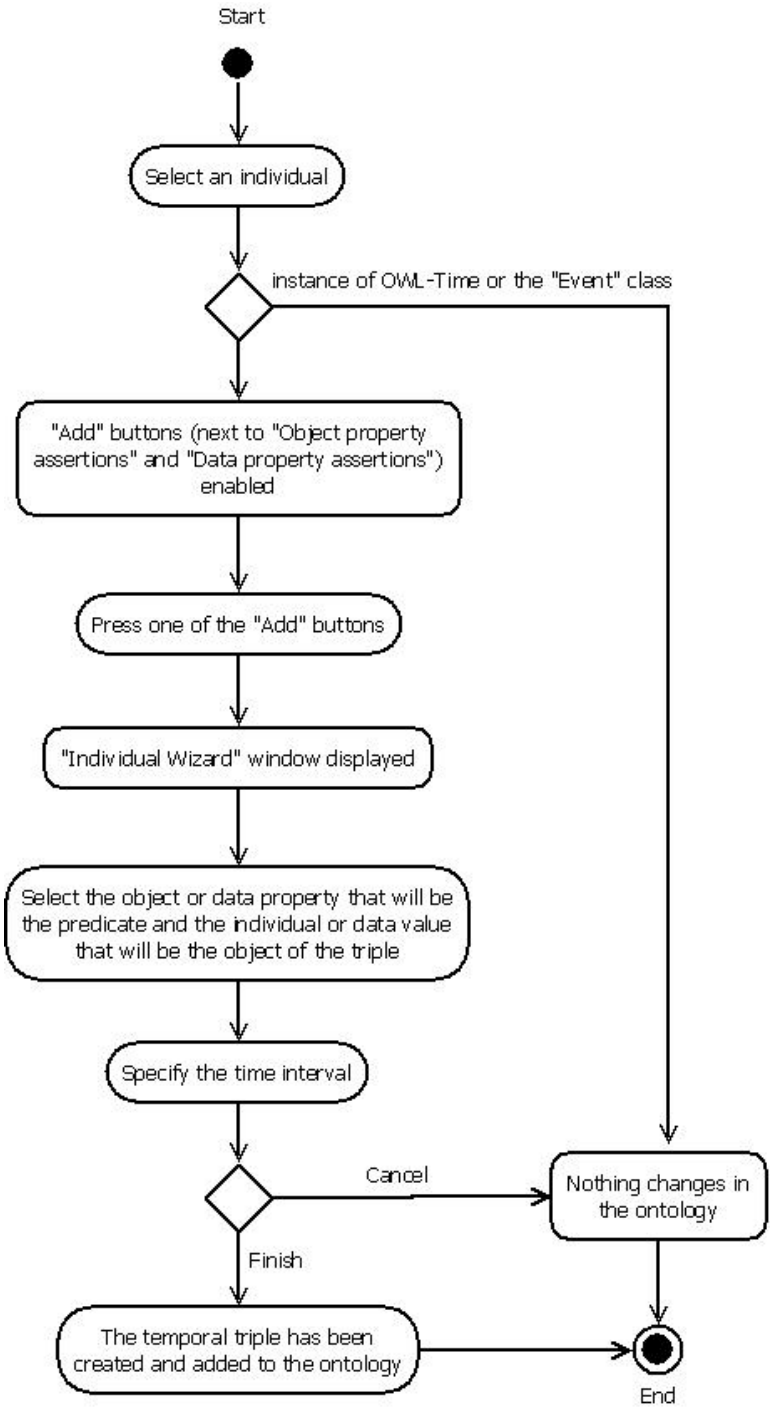


Figure 4.8: Activity Diagram 8. Add a temporal property assertion.

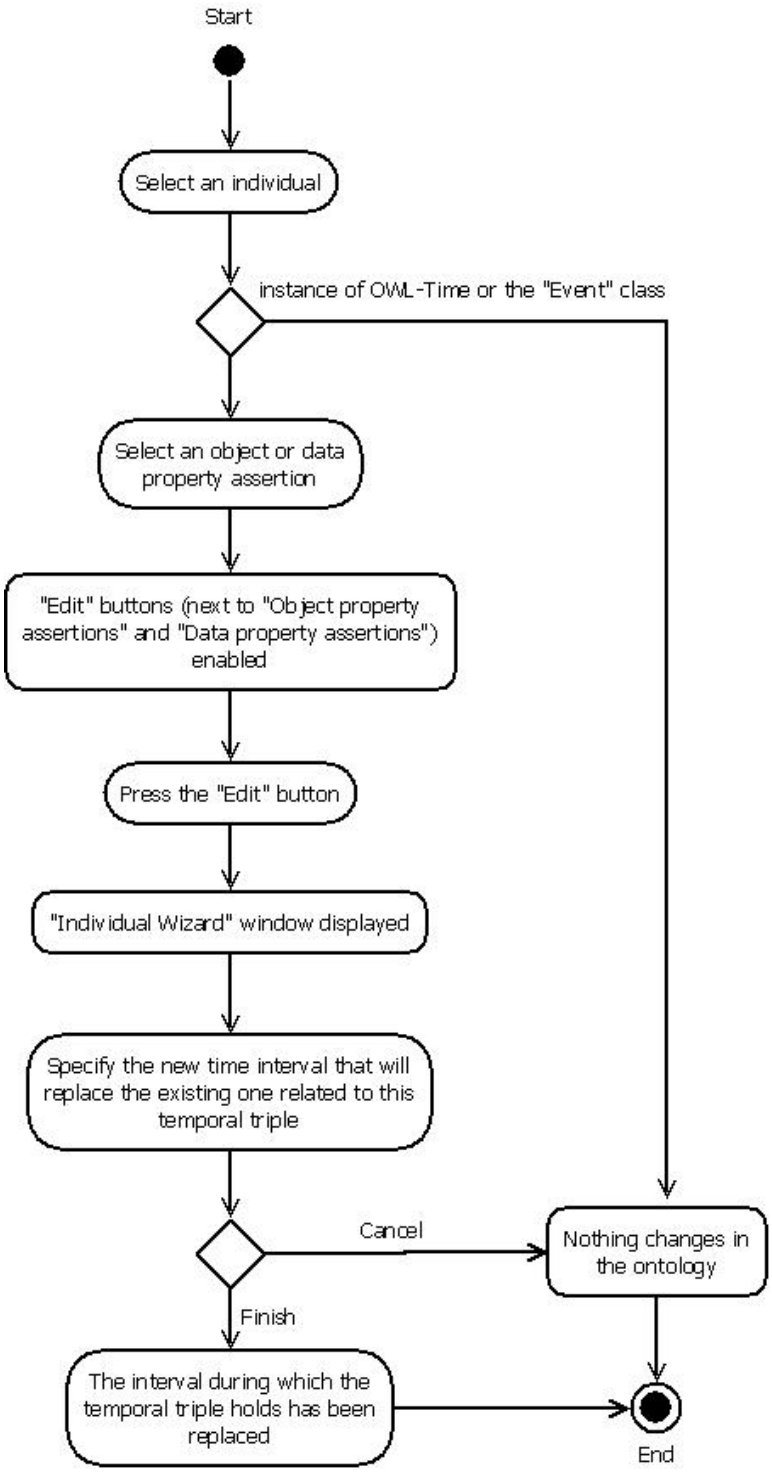


Figure 4.9: Activity Diagram 9. Edit the interval of a temporal property assertion.

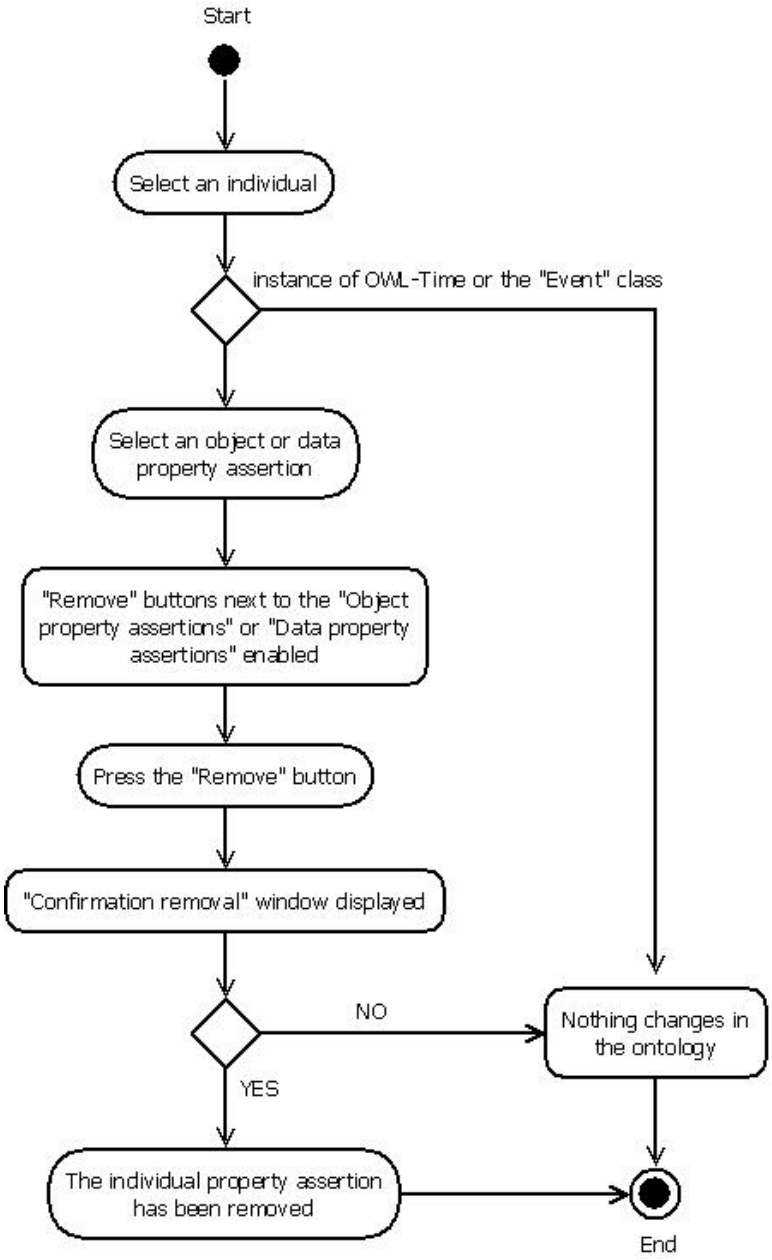


Figure 4.10: Activity Diagram 10. Remove a property assertion.

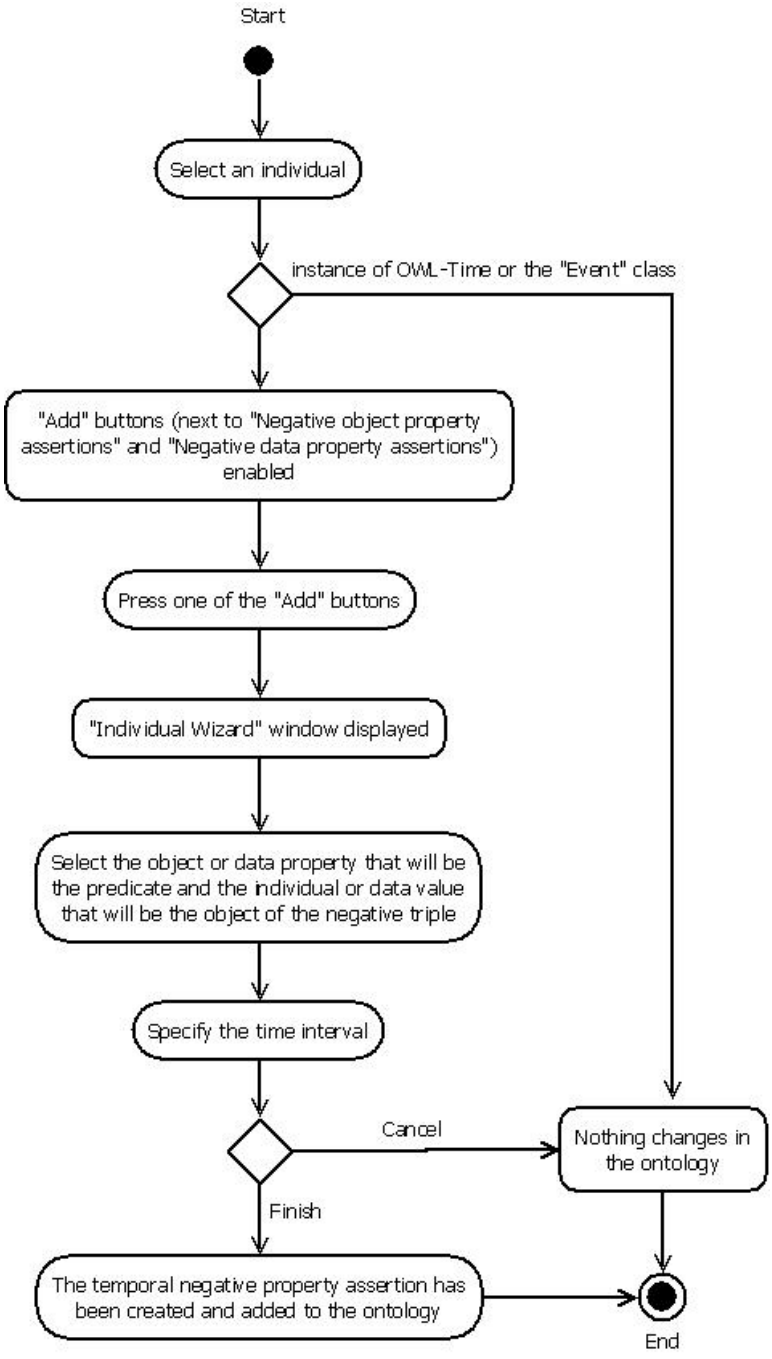


Figure 4.11: Activity Diagram 11. Add a temporal negative property assertion.

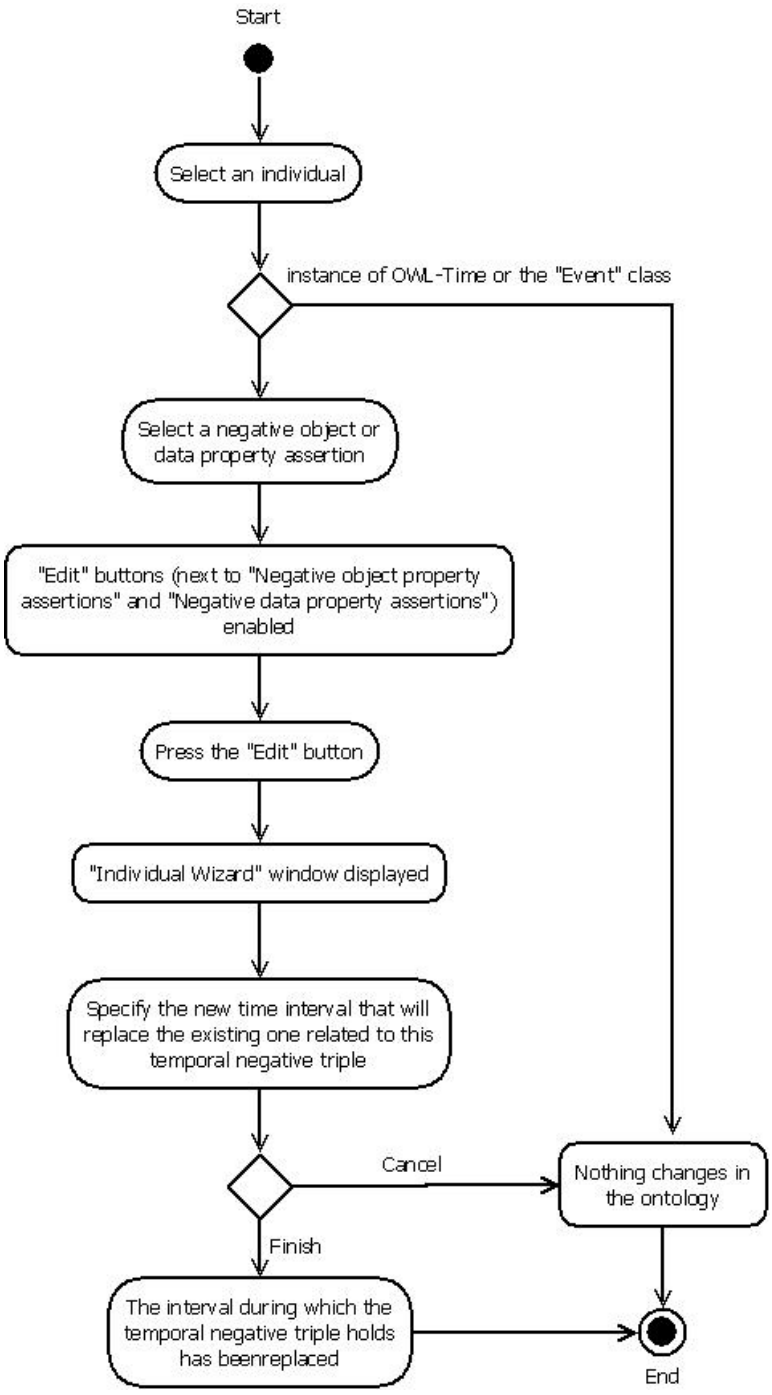


Figure 4.12: Activity Diagram 12. Edit the interval of a temporal negative property assertion.

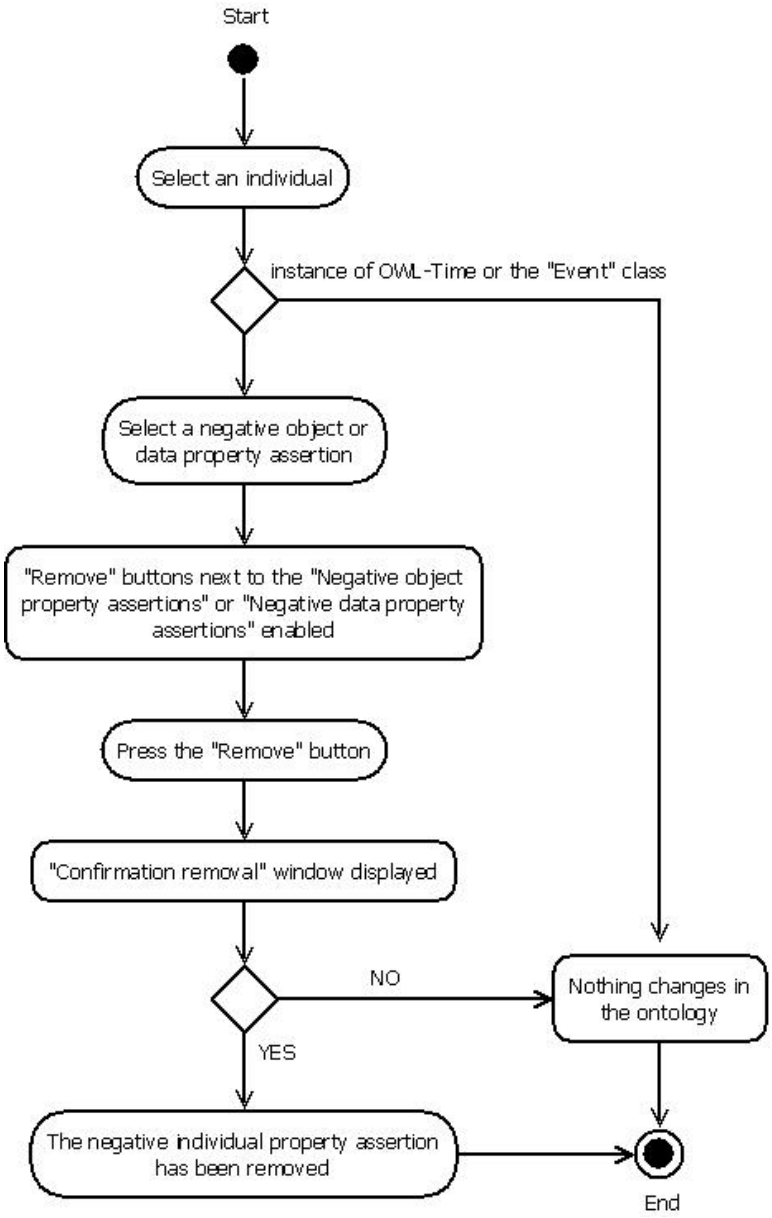


Figure 4.13: Activity Diagram 13. Remove a negative property assertion.

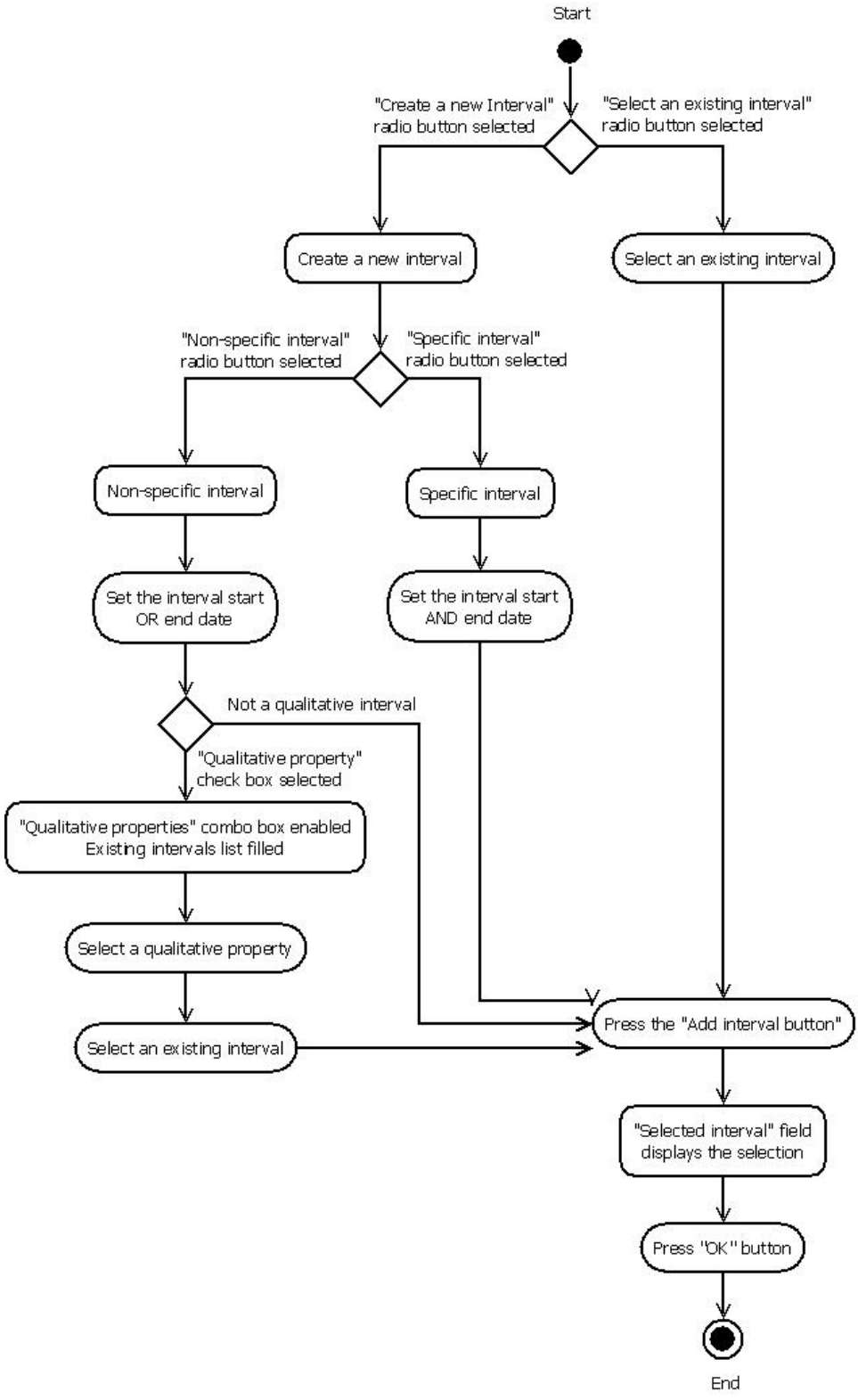


Figure 4.14: Activity Diagram 14. Create a time interval.

## 4.3 User Interface Design

The interface layout used in Chronos was originally created by Polyxeni Makri in her work on *4D-Fluents Plug-in* [21]. Several improvements and modifications have been made though, in order to serve the purposes of our tool. The layout was inspired from the structure of standard Protégé tabs. This way, users of Protégé will not experience any difficulties in using Chronos.

We provide four different views on the left of the screen, one for each entity type. These are the "Class hierarchy", the "Object property hierarchy", the "Data property hierarchy" and the "Individuals by type" views. The main Chronos' panel is on the right of the screen and it is called "Chronos View Component". The user can select a class, an object property, a data property or an individual from the views on the left and "Class", "Object property", "Data property" or "Individual" panel will be be displayed respectively, in "Chronos View Component".

The north part of "Chronos View Component" is common in all four panels. It contains the buttons for the conversion of the entity (and in the case of class, the button for the creation of a new instance), a help text that guides the user through the selections and an "Info" panel that provides useful information about the selected entity. The rest of the view differs depending on the type of the entity.

In the case that the user selects a class from the "Class hierarchy" view, the class panel will be displayed in "Chronos View Component" and the layout will look like that illustrated in figure 4.15. In the center of the view there are two lists. One that displays the equivalent classes of the selected class and one that displays its superclasses. At the right of each list, there are three button that manipulate it.

If an object property is selected, the "Chronos View Component" will display the "Object property" panel that is illustrated in figure 4.16. The central layout is designed according to the default Protégé object property tab. It includes the "Object property characteristics", which is a list of check-boxes that indicate whether the selected object property is *functional*, *inverse functional*, *transitive*, *symmetric*, *asymmetric*, *reflexive* and *irreflexive*. The "Object property description" contains information about the property's domain, range, equivalent properties, super properties, inverse properties and disjoint properties. There is also a panel at the bottom of our view, which expands at the click of the "Show" button, and illustrates the representation of the selected object property.

Similarly, in the case of a selected data property, the "Data property" panel is illustrated in figure 4.17. It also includes the "Data property characteristics" panel, which contains a check-box that indicates whether the selected data property is *functional*. The "Data property description" panel contains information about the property's domain, range, equivalent prop-

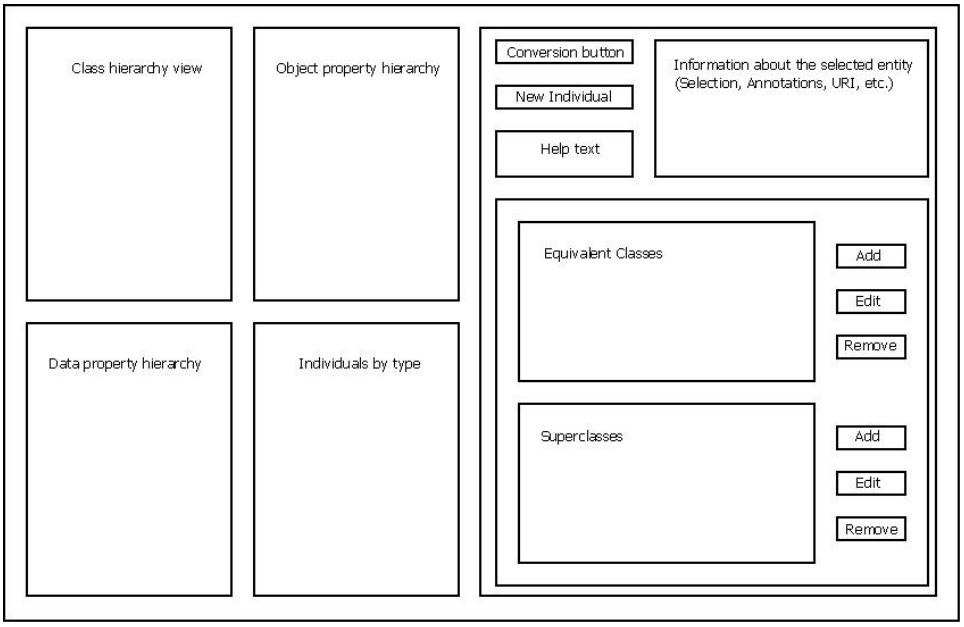


Figure 4.15: Class panel layout

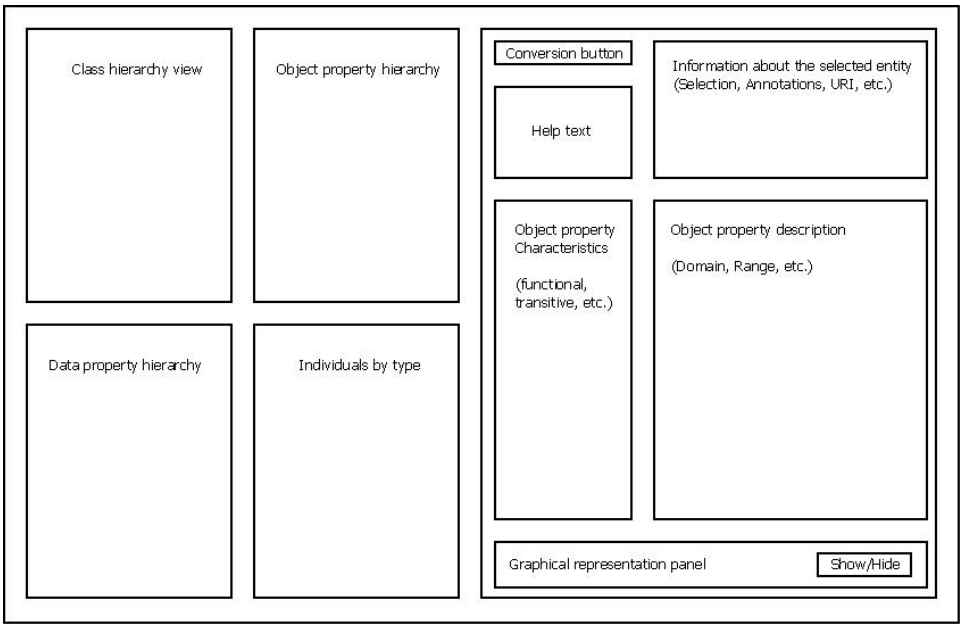


Figure 4.16: Object property panel layout

erties, super properties and disjoint properties. Just as in object properties, there is a graphical representation panel at the bottom of the "Data property" panel, too.

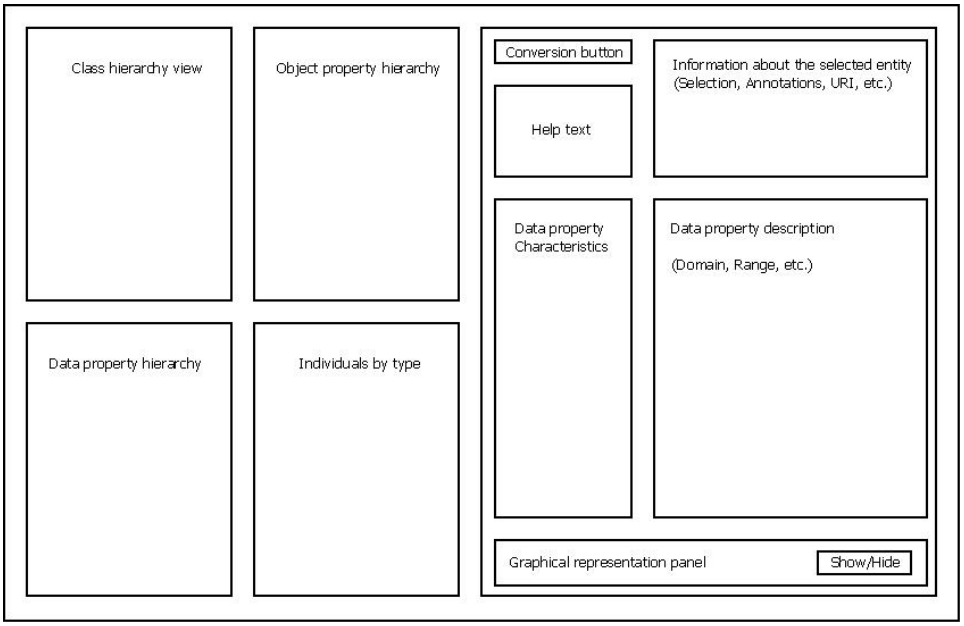


Figure 4.17: Data property panel layout

Finally, when the user selects an individual, the "Individual" panel will be displayed. It is also designed according to the default Protégé's individual panel. On its left side there is the "Description" panel, which includes three lists. The first displays the types of the selected individual, the second displays the individuals that are declared to be the same with the one selected and the third displays those individuals that are declared to be different from the one selected. On the right side of the "Individual" panel there is the "Property assertions" panel. It includes four lists, one for the object property assertions, one for the data property assertions, one for the negative object property assertions and one for the negative data property assertions. On the right side of each of these lists there are three buttons used for manipulating them. The layout of the "Individual" panel is illustrated in figure 4.18.

## 4.4 Code Structure

The source code that implements Chronos was organized in several packages, in order to make it more flexible and easily expandable. The classes that implement the graphic user interface were stored in different packages to

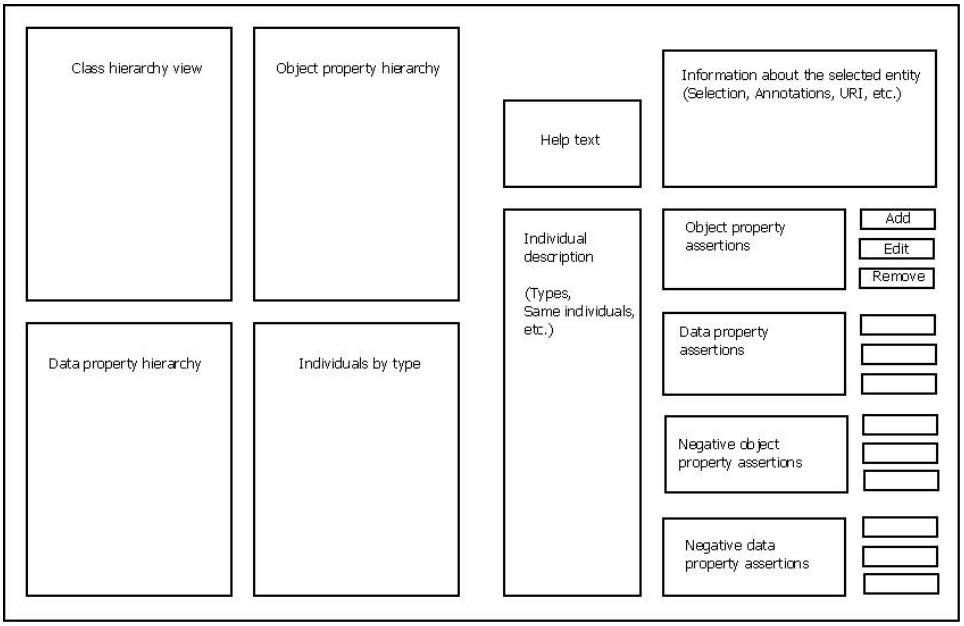


Figure 4.18: Individual panel layout

those that implement the temporal representation model. More specifically the packages used are:

**classPanels, objectPropertyPanels, dataPropertyPanels, individualPanels** they contain classes that implement panels that are included in the main window of Chronos and the dialog windows that are created. As explained in the previous section, for each different type of entity selected, a different panel is displayed. Each one of these packages contain the basic components that combine to create those panels. For instance, the *classPanels* package contains all the classes that take part in the creation of the "Class" panel illustrated in figure 4.15.

**structures** contains classes whose main purpose is to store and distribute data between other classes. It includes data structures that hold the participants of each temporal constraint, temporal triples or temporal negative triples. For example, an instance of the class *TemporalObjectPropertyTriple* stores the subject and object individual, the event individual, the interval individual during which the temporal triple holds and the object property that is the predicate of the triple, and provides the setters and the getters methods to access that data.

**tab** it only contains the class that implement the Chronos tab.

**time** this package contains two classes. The *TimeFactory* class, which contains the methods used for the conversion of the entities, and the

*ConstraintFactory* that creates the temporal constraints. The *TimeFactory* class is the most frequently accessed class, and it is called almost from every other class in our implementation. The *ConstraintFactory* is only accessed by the *TimeFactory*.

**view** contains the top level class *ChronosViewComponent*, the *MergeDialog* class that implements a dialog window that appears when clicking on Chronos tab for the first time, a *Renderer* class and the *Updatable* interface that is implemented by all the panel classes.

Figure 4.19 illustrates the class diagram of the source code of our application.

## 4.5 Plug-in Documentation

As we have already mentioned, all the methods that are used for adding the dimension of time in the active ontology are included in the *TimeFactory* class. In this section, we present the most important methods that are used for implementing the N-ary relations.

- `boolean containsOWLTime(OWLOntology)` is a boolean method that iterates over the entities of the ontology that is given as an argument looking for the namespace of the OWL-Time ontology. If there are entities that start with that namespace, they are OWL-Time entities and the method return true.

- `createRequiredEntities()` As explained in section 3.2, there are some entities that are required in order to implement the N-ary relation representation. This void method inserts these entities in the ontology.

- `isTemporalProperty(OWLEntity)` is a boolean method that returns true if the entity given as an argument is a temporal object or data property, false otherwise. What this method actually does is to check the property's domain and range whether or not it is a union of some class expression and an Event class. In the case of a data property, only the domain is checked.

- `convertObjectPropertyToTemporal(OWLObjectProperty, OWLNamedIndividual)` is a void method that converts the given object property to temporal. The property's domain and range will be changed as explained in 3.2.1. Then two other methods are called, the `convertObjectPropertyTriples` and the `applyConstraintsOnObjectproperty`, which are explained below.

- `convertObjectPropertyTriples(OWLObjectProperty, OWLNamedIndividual)` is a method that iterates over the object property and the negative

Figure 4.19: The class diagram of the source code

object property axioms that contain the given object property and converts them according to the N-ary relations model, as described in 3.2.3. The given individual is the time interval individual during which all the temporal triples hold. If it is *null*, each triple will hold for a (different) unknown time interval.

- `convertDataPropertyToTemporal(OWLDataProperty, OWLNamedIndividual)` similarly to object properties, this method converts the given data property to temporal as described in 3.2.2. Afterwards, it calls the `convertDataPropertyTriples` and the `applyConstraintsOnDataproperty`, which are explained below.

- `convertDataPropertyTriples(OWLDataProperty, OWLObjectProperty, OWLNamedIndividual)` is a method that iterates over the data property and the negative data property axioms that contain the given data property and converts them according to the N-ary relations model. The given individual is the time interval individual during which all the temporal triples hold. If it is *null*, each triple will hold for a (different) unknown time interval.

- `createTemporalObjectPropertyAxiom(OWLClassExpression, OWLNamedIndividual, String, OWLObjectProperty, OWLNamedIndividual, Sting, String, String, OWLNamedIndividual)` is a method that creates a new individual and all the axioms needed to connect it to an already existing individual with a temporal object property.

- `createTemporalDataPropertyAxiom(OWLClassExpression, OWLNamedIndividual, String, OWLDataProperty, String, Sting, String, String, OWLNamedIndividual)` is a method that creates a new individual and all the axioms needed to connect it to an already existing individual with a temporal data property.

- `createInterval(String, String, String, OWLNamedIndividual)` is a method that return the OWLNamedIndividual that represents a time interval. This interval can be specific or non-specific, depending on the adequacy of the given arguments.

- `applyConstraintsOnObjectProperty(OWLObjectProperty)` adds the SWRL rules and the axioms needed in order to convert to temporal the object property restrictions of the given object property, as described in 3.3.

- `applyConstraintsOnDataProperty(OWLDataProperty)` adds the SWRL rules and the axioms needed in order to convert to temporal the data property restrictions of the given data property, as described in 3.3.

- `convertClassToTemporal(OWLClass, OWLNamedIndividual)` iterates over the properties that are related to the given class and depending on the type of the property, it calls the `convertObjectPropertyToTemporal` or the `convertDataPropertyToTemporal` method. The given individual is the time interval and it is passed to the called methods.

- `id()` returns a long integer that is used for giving different names to the entities that are created by our application (individuals of the type Interval, Event, etc.). The number returned is the current time in milliseconds, possibly incremented.

There are many other methods in *TimeFactory* that are called by those presented, but most of them are utility methods, or are created to simplify the code.

The *ConstraintFactory* class contains methods that return sets of OWL axioms or SWRL rules, which implement the temporal version of the property restrictions described in 3.3.

# Chapter 5

# Conclusion and Future Work

We introduce Chronos, a tab plug-in for Protégé editor that facilitates the creation and editing of temporal OWL 2.0 ontologies. The temporal concepts as well as the properties that evolve over time are represented by means of the N-ary Relations model [5], that is a W3C recommendation. Chronos enables the use of restrictions on temporal properties, which have different semantical meaning than those applied on static properties. The user does not have to be familiar with the peculiarities of the temporal representation model, thus making the manipulation of temporal entities as easy as if they were static.

Extending Chronos to include a query language that supports querying over temporal ontologies is an interesting issue for future work.

# Bibliography


[1] Deborah L. McGuinness, Frank van Harmelen: *"OWL Web Ontology Language Overview"*. W3C Recommendation, February 2004. http://www.w3.org/TR/owl-features/

[2] Michael K. Smith, Chris Welty, Deborah L. McGuinness: *"OWL Web Ontology Language Guide"*. W3C Recommendation, February 2004. http://www.w3.org/TR/owl-guide/

[3] Jerry R. Hobbs, Feng Pan: *"Time Ontology in OWL"*. W3C Recommendation, September 2006. http://www.w3.org/TR/owl-time/

[4] C. Welty and R. Fikes: *"A Reusable Ontology for Fluents in OWL"*. Frontiers in Artificial Intelligence and Applications, 150:226236, 2006.

[5] Natasha Noy, Alan Rector: *"Defining N-ary Relations on the Semantic Web"*. W3C Working Group Note, April 2006. http://www.w3.org/TR/swbp-n-aryRelations/

[6] http://protege.stanford.edu/

[7] B. Parsia, E. Sivrin: *"Pellet: an OWL-DL Reasoner"*. ISWC 2004 Proceedings. http://clarkparsia.com/pellet/protege/

[8] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof and M. Dean *"SWRL: A Semantic Web Rule Language Combining OWL and RuleML"*. W3C Member submission, 2004. http://www.w3.org/Submission/SWRL/

[9] Sotiris Batsakis, Euripides G.M. Petrakis: *"Representing Temporal Knowledge in the Semantic Web: The Extended 4D Fluents Approach"*, 2nd International Workshop on Combinations of Intelligent Methods and Applications (CIMA' 2010), October 2010.

[10] Sotiris Batsakis, Euripides G.M. Petrakis: *"SOWL: Spatio-temporal Representation, Reasoning and Querying over the Semantic Web"*, 6th International Conference on Semantic Systems (I-SEMANTICS' 2010), September 2010.

[11] Thomas C. Jepsen: *"Just What Is an Ontology, Anyway?"*, IT Pro September/October 2009, Published by the IEEE Computer Society, 1520-9202/09/26.00 2009 IEEE.

[12] A. Johannes Pretorius: *"Ontologies - Introduction and Overview"*, Adapted from: PRETORIUS, A.J., Lexon Visualisation: Visualising Binary Fact Types in Ontology Bases, Chapter 2, Unpublished MSc Thesis, Brussels, Vrije Universiteit Brussel, 2004.

[13] http://www.w3.org/TR/owl2-overview/

[14] J. F. Allen: *"Maintaining Knowledge About Temporal Intervals"*. Communications of the ACM, 26:832-843, 1983.

[15] A. Artale, E. Franconi: *"A Survey of Temporal Extensions of Description Logics"*, Annals of Mathematics and Artificial Intelligence, 30(1-4), 2001.

[16] C. Lutz, F. Wolter, M. Zakharyaschev: *"Temporal Description Logics: A Survey"*, In Proc. TIME08, IEEE Press, 2008.

[17] M. Klein, D. Fensel: *"Ontology Versioning for the Semantic Web*, In International Semantic Web Working Symposium (SWWS01), pages 7592, California, USA, JulyAugust 2001.

[18] Matthew Horridge, Sean Bechhofer: *"The OWL API: A Java API for Working with OWL 2 Ontologies"*, The University of Manchester, UK.

[19] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, Lynn Andrea Stein: *"OWL Web Ontology Language Reference"*, W3C Recommendation, February 2004
http://www.w3.org/TR/owl-ref/

[20] Matthew Horridge, Peter F. Patel-Schneider: *"OWL 2 Web Ontology Language Manchester Syntax"*, W3C Working Group Note, October 2009.
http://www.w3.org/TR/owl2-manchester-syntax/

[21] Polyxeni Makri: *4D-Fluents Plug-In: A Tool for Handling Temporal Ontologies in Protégé*, Diploma Thesis, Department of Electonic and Computer Engineering, Technical University of Crete, May 2011